SAIC-94/1142

Global Association
Design Document and User's Manual

R. Le Bras, H. Swanger, T. Sereno, G. Beall, R. Jenkins and W. Nagy

November 17, 1994

Science Applications International Corporation
10260 Campus Point Drive
San Diego, California 92121

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	23 November 1994	Technical Report

4. TITLE AND SUBTITLE

Global Association Design Document and User's Manual

5. FUNDING NUMBERS

F08606-90-D-0005

6. AUTHOR(S)

R. LeBras, H. Swanger, T. Sereno, G. Beall, R. Jenkins and W. Nagy

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Science Applications International Corporation
10260 Campus Pt. Drive
San Diego, CA 92121

8. PERFORMING ORGANIZATION REPORT NUMBER

SAIC-94-1142

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

HQ Air Force Technical Applications Center
(HQ AFTAC/TTR)
1030 S. Highway A1A
Patrick AFB FL 32925-3002

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release, Distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This document describes the *Global Association System* for automatic interpretation of seismic data to associate signals and locate seismic events. This system uses a hybrid method that integrates techniques in generalized beam forming [e.g., *Ringdal and Kværna*, 1989; *Taylor and Leonard*, 1992; *Leonard*, 1993] and expert systems [*Bache et. el.*, 1993]. It is designed to handle the large volumes of data that are needed to monitor compliance with a Comprehensive Test Ban Treaty (CTBT).

14. SUBJECT TERMS

Global Association
DEsign Document
GA
User's Manual
StaPro
SAIC-94-1142

15. NUMBER OF PAGES

67

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	Same as Report

(This page intentionally left blank.)

Table of Contents

List of Figures

Page

List of Tables

1.0 Introduction

This document describes the *Global Association System* for automatic interpretation of seismic data to associate signals and locate seismic events. This system uses a hybrid method that integrates techniques in generalized beam forming [e.g., *Ringdal and Kværna*, 1989; *Taylor and Leonard*, 1992; *Leonard*, 1993] and expert systems [*Bache et. el.*, 1993]. It is designed to handle the large volumes of data that are needed to monitor compliance with a Comprehensive Test Ban Treaty (CTBT).

1.1 Background

In 1993, discussions began concerning the design of the International Data Center (IDC) and U.S. National Data Center (NDC) for the upcoming Group of Scientific Experts Third Technical Test called GSETT-3 (for an overview, see *Kerr* [1993]). The GSETT-3 experiment will be the first demonstration of a global monitoring system that addresses the CTBT problem. In the early discussions, the global network was envisioned to include as many as 60 primary stations (mostly arrays) to provide continuous data, and up to 200 secondary stations to provide waveform segments upon request. The volume of data (~10 Gbytes per day) and number of events (300-400 per day) were estimated to be approximately a factor of five to ten times greater than encountered from existing global networks.

SAIC performed an engineering study to assess whether existing seismic monitoring systems could be modified to handle these expected data volumes. The software components of the ADSN and IMS were considered. The conclusion of this study was that the primary bottleneck would be the automatic association and location program, **ESAL** [*Bratt et al.*, 1991, 1994]. Performance analyses indicated that **ESAL**'s execution time scaled roughly with the square of the detection density. This was unacceptable since extrapolation to the estimated data volumes for a CTBT monitoring network indicated that **ESAL** would not be able to process the data in real time.

To address this concern, SAIC proposed to replace **ESAL** with a new hybrid method. This method splits the tasks currently performed by **ESAL** into separate modules that can be run in parallel (Figure 1). The main association module is very similar to published generalized beam forming techniques and to the unpublished technique used by the Australian IDC in the GSETT-2 experiment [Ken Muirhead, personal communication]. The main difference is that a model of the probability of detection is used to significantly reduce the search space. Initial work on the new hybrid method was supported by the Advanced Research Projects Agency (ARPA). That work was continued under this AFTAC task order.

1.2 Report Outline

This document contains descriptions of the individual components of the *Global Association System*. Included are descriptions of the algorithms, data flow, and a summary of the major components. The system currently includes **ESAL** for conflict resolution and event refinement. **ESAL** has been described in detail by *Bratt et al.* [1991, 1994], and that level of detail will not be repeated here.

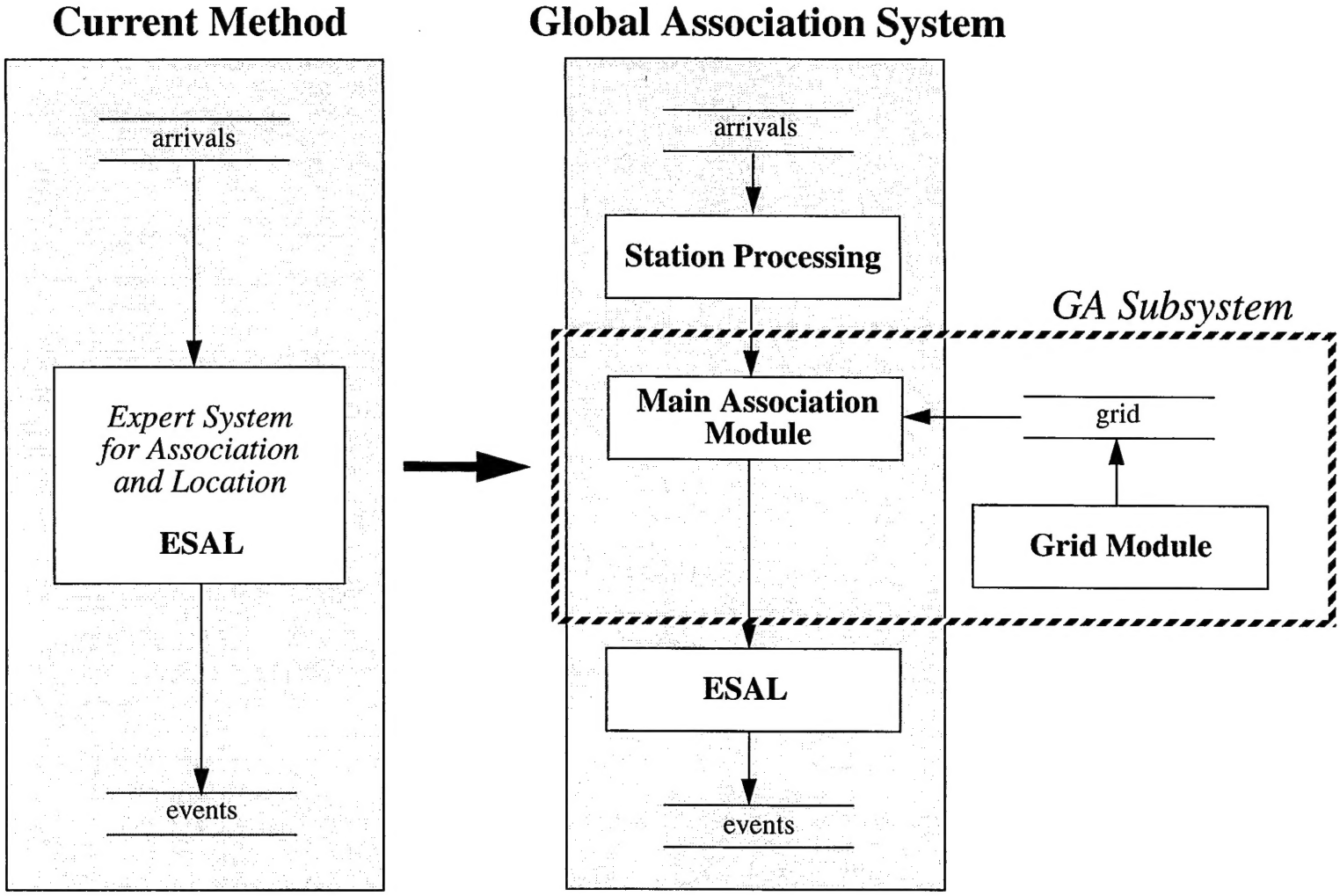


Figure 1. The current and planned configuration of the software for automated association and location are shown.

2.0 System Summary

The main components of the *Global Association System* are:

- **StaPro** - This module performs station processing. It analyzes detections and their features to make preliminary seismic phase identifications. **StaPro** contains the same logic as **ESAL** for this task, and it includes the ability to compute single-station location and magnitude hypotheses. This information is used by **GAassoc** to screen detections from local events with magnitudes less than a user-specified threshold. This module is considerably more compact than **ESAL**, permitting each station to be processed independently and in parallel.

- **GAcons** - This module builds a global grid file containing the knowledge base for the association process. Overlapping circular grid cells provide complete global coverage, including depth cells in areas where deep seismicity is known to occur. The information contained in the grid file includes travel time, slowness, and azimuth bounds, and information on the probability of detection for each station in the network. A graphical user-interface is available to review and edit the grid values.

- **GAassoc** - This module identifies event hypotheses using an exhaustive search over all grid cells. It uses the information in the grid file produced by **GAcons** to identify detections that are consistent with a particular event hypothesis. The preliminary bulletin may contain conflicting associations (i.e., phases that are associated to more than one event hypothesis). These conflicts are resolved by **ESAL** (see below).

- **EServer/ESAL** - These modules are used to resolve conflicts and refine the event hypotheses formed by **GAassoc**. **EServer** is a data agent for **ESAL** that prepares ASCII input files from data read from a commercial relational database management system (RDBMS). **EServer** also writes events and associations determined by **ESAL** to the RDBMS.

The high-level processing and data flow are shown in Figure 2. **GAcons** is not included because it is not part of the real-time processing. It produces a static grid file that must be recomputed only if the network changes or modifications are made to the knowledge base. **StaPro** is triggered by the completion of signal detection and feature extraction for a particular station. **GAassoc** performs the global association after **StaPro** has completed for all stations. Information about detections are transferred from the individual **StaPro** runs to **GAassoc** through the RDBMS using the CSS 3.0 database schema [*Anderson et al.*, 1990]. **GAassoc** writes all event hypotheses and associations to the RDBMS. **EServer** reads these hypotheses, prepares the ASCII input data, and initiates **ESAL** to resolve conflicts, refine event hypotheses, and produce the final bulletin. Finally, **EServer** writes **ESAL**'s results to the RDBMS.

The Task Controller in Figure 2 could be an automated or manual process. In the simplest configuration, the Task Controller is a user that manually initiates each component after the previous one has completed. In the ADSN and IMS, the Task Controller is a separate software module called the **Process Manager** [*Given et al.*, 1993]. However, the Task Controller does not need to initiate **ESAL** since this is done automatically by **EServer**.

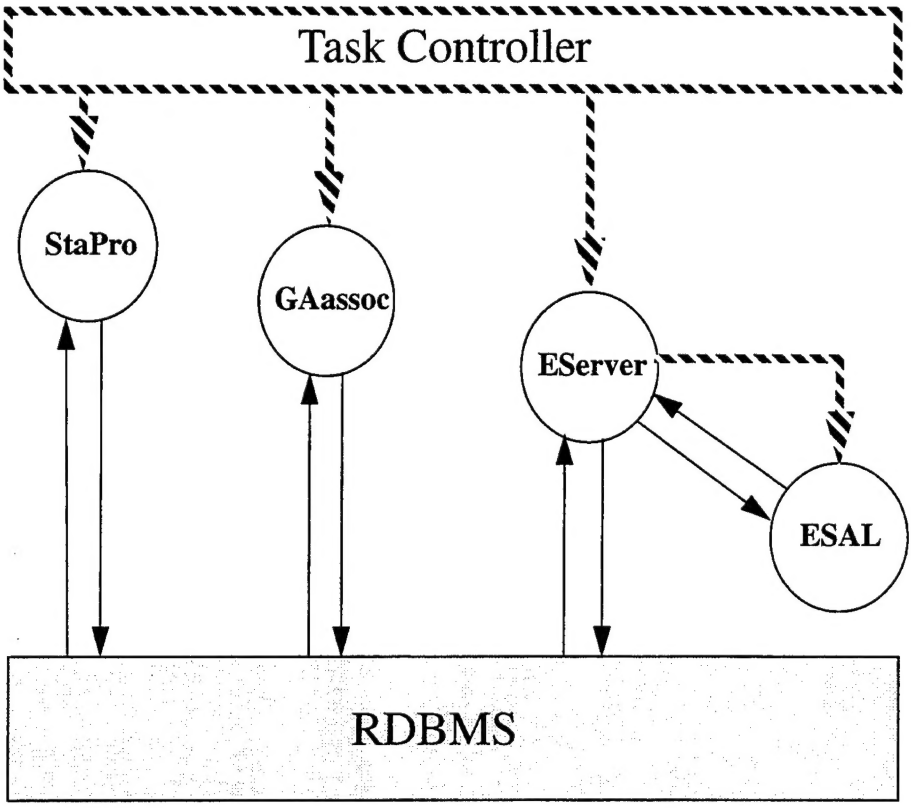


Figure 2. This shows the high-level processing and data flow for the *Global Association System*. Process flow is indicated by dashed lines, and data flow is indicated by solid lines.

The *Global Association System* was developed on UNIX workstations under the Solaris 2.3 Operating System. The current version uses an Oracle™ 7.1.3 RDBMS. **StaPro** uses CLIPS Version 6.0 to provide run-time configurability of station-specific rules. CLIPS is a knowledge-based macro language which is supported by NASA. **ESAL** is programmed in the ART (Automated Reasoning Tool) expert system shell from Inference Corporation.

3.0 System Components

This section describes each major component of the *Global Association System*. For each new component (**StaPro**, **GAcons**, and **GAassoc**), we provide a summary of the algorithms that are used, the data flow, and a summary of the major software components. **ESAL** has already been documented in detail by *Bratt et al.* [1991, 1994]. Therefore, Section 3.3 describes only the **ESAL** tasks that are relevant for the *Global Association System*. The format of this description is consistent with the existing **ESAL** documentation. The appendices give detailed descriptions of the user-parameters for each module.

3.1 Station Processing (StaPro)

StaPro determines the initial wave type of each detection (Teleseism, Regional *P*, Regional *S*, or Noise), groups detections that appear to come from the same event, and determines a preliminary identification of the seismic phase (*P*, *Tx*, *Pn*, *Pg*, *Px*, *Sn*, *Lg*, *Rg*, *Sx*, or *N*). In addition, single-station location and magnitude are computed for groups that pass user-specified event confirmation criteria. The major steps are illustrated in Figure 3.

Station processing is done by **ESAL** in the current IMS and ADSN systems. **StaPro** was written in C and CLIPS to replace **ESAL**'s station processing. The main advantages:

- *Improved station characterization*
  Parameters and rules can be customized for each station.

- *Improved processing speed*
  Tests have shown that **StaPro** runs nearly three times faster than **ESAL**'s station processing to accomplish the same task.

- *Lower operating costs*
  StaPro uses CLIPS for knowledge-based processing which is available for a one-time nominal fee. **ESAL** requires a separate ART license for each machine.

3.1.1 Algorithms

This section summarizes the main algorithms used in **StaPro**. More detailed descriptions are provided by *Bratt et al.* [1991, 1994]. In the current version, CLIPS is only used for initial wave-type identification. We plan to extend **StaPro** to also use CLIPS to implement station-specific rules for phase identification. The following subsections refer to user-parameters that are described in Appendix A. We use the following distinct font and style to identify these parameters: *user-parameters*.

Initial Wave-Type Identification

The first task is to identify the initial wave type of each detection as *T* (teleseism), *P* (regional *P*), *S* (regional *S*) or *N* (noise). **StaPro** allows station-specific rules (SSR) to be specified to supplement or override the default method. The default method is the same as the one used by **ESAL** [*Bratt et al.*, 1991, 1994].

Station-Specific Rules. If station-specific rules are specified by the user (see the *sta-rule-file* parameter), then they are applied before the default rules. If the return status is satisfactory and the initial wave type is valid, then **StaPro** will proceed to the next detection. Otherwise, the default method is used to determine the initial wave type (see below).

The SSRs are written in CLIPS and should either confirm (phase P) or restrict (not-p) one of the valid wave types. In either case the conclusion must be asserted on the CLIPS fact list to make it

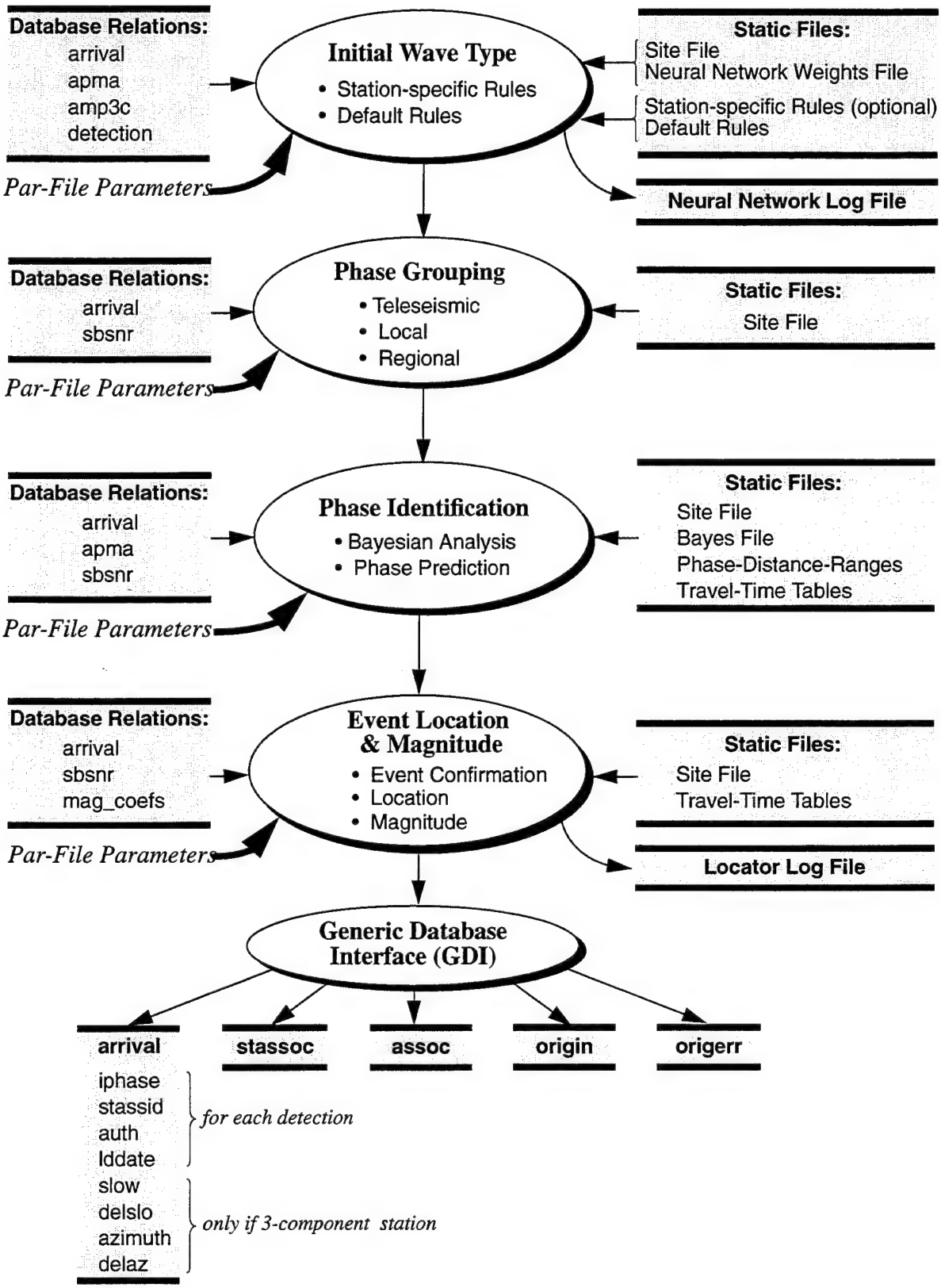


Figure 3. This is a data flow diagram for **StaPro**.

available for subsequent reasoning. The following is an example of an SSR that will restrict the wave-type from being identified as *P* if the rectilinearity is less than a user-defined threshold:

```
;;; wave type cannot be P if rect < threshold
(defrule Sample-Sta-Rule-Not-P                ;; comments
   (program-state ssr-initial-wave-type)      ;; required for IWT rules
   (rect ?r)                                  ;; rectilinearity from the RDBMS
   (min-p-rect ?mpr)                          ;; user-defined threshold
   (test (check-rect-range ?r))               ;; rectilinearity value is valid
   (test (<= ?r ?mpr)                         ;; rectilinearity <= threshold
=>
   (assert (status 1))                        ;; status 1
   (assert (not-p)))                          ;; wave type is not P
```

The left-hand side (LHS) of the rule consists of conditional elements, and the right-hand side (RHS) consists of actions. An arrow ("=>") separates the LHS from the RHS. All conditions on the LHS must be met before actions on the RHS will be executed. This example asserts (not-p) on the fact list to restrict this detection from being identified as a regional *P*-wave. If no other SSRs are fired, then the default rules will be used to identify the initial wave type subject to the constraint that it is not *P*. Similar restrictions can be applied for any of the valid wave types.

The next example is slightly more complicated. It is a rule to set the wave type to *T* if the rectilinearity is above a user-threshold and the period is less than a user-threshold.

```
;;; wave type is T if rectilinearity is high and period is low
(defrule Sample-Sta-Rule-T                    ;; comments
   (program-state ssr-initial-wave-type)      ;; required for IWT rules
   (rect ?r)                                  ;; rectilinearity from the RDBMS
   (min-t-rect ?mtr)                          ;; user-defined threshold
   (test (check-rect-range ?r))               ;; rectilinearity value is valid
   (test (>= ?r ?mtr)                         ;; rectilinearity >= threshold
   (per ?p)                                   ;; period from the RDBMS
   (max-t-per ?mtp)                           ;; user-defined threshold
   (test (check-per-range ?p))                ;; period value is valid
   (test (<= ?p ?mtp)                         ;; rectilinearity <= threshold
=>
   (assert (status 1))                        ;; status 1
   (assert (phase T)))                        ;; wave type is T
```

All SSRs must contain (program-state ssr-initial-wave-type) as a conditional element. This will ensure proper processing flow of the rules. In addition, at least one of the rules must contain the action which summarizes the overall completion status for the SSRs. This is done by placing (status 0 | 1) on the RHS. A status of 1 represents satisfactory completion, and a status of 0 means that completion was not satisfactory. Many helpful examples of SSRs exist in the default rule file called StaPro-IWT.clp which is included in the **StaPro** software release. It can be copied

and modified to meet station-dependent requirements. The CLIPS Reference Manual should be referred to for further details.

Default Method. The default method of determining initial wave-type is two-tiered. A neural network will be used if weights are available for the station in question (see the *nnet-weights-file* parameter). Otherwise, default rules will be used. The neural network is preferable for three-component stations since unreliable slowness estimates make it difficult to distinguish between *P* and *S* waves. *Sereno and Patnaik* [1993] provide a detailed description of the three-stage neural network that is used in **StaPro** for initial wave-type identification.

The default rules are used if neural network weights are not found or if the neural network returns an error status. They are guaranteed to return one of the four initial wave-types. The rules are different for arrays than they are for three-component stations. For arrays the initial wave type is based on the *f-k* slowness, and for 3-component stations it is based on polarization attributes and frequency content [*Bratt et al.*1991, 1994].

The last stage of initial wave-type identification is the possible revision of *T-to-P* if certain criteria are met. This is not considered until all detections have been assigned an initial wave-type. Revision of *T-to-P* can only occur when a compatible regional *S* arrival is found and *P* is an allowed wave type. If the station is an array, then the velocity from *f-k* analysis must also be less than the minimum teleseismic velocity (see the *min-teleseism-velocity* parameter). Details are provided by *Bratt et al.* [1991, 1994].

Phase Grouping

The second major task in station processing is phase grouping. **StaPro** uses exactly the same method as **ESAL** for this task. A brief description is given below, and details are provided by *Bratt et al.* [1991, 1994].

After all detections have been assigned an initial wave type, they are collected into groups that appear to come from the same event. The first arrival in each group is called the *generator*, and its initial wave type must be either *P* or *T*. Subsequent arrivals are added to the group on the basis of their compatibility (based on azimuth and amplitude) with the *generator*. All members of a group are assigned the same stassid value (stassid is a unique key in the RDBMS to identify station association groups; see *Anderson et al.* [1990]). Noise detections are assigned a stassid of -1 indicating they don't belong to any association group. *Generators* with no compatible detections are also assigned a stassid of -1. Three categories of events are treated separately in phase grouping: teleseismic, local and regional. Teleseismic and local groups are found first, and their detections are removed from consideration for regional grouping. All regional *P*-wave detections that were not previously assigned to a local group are potential *generators* for a regional group. Each search uses special compatibility parameters which define limitations for that respective category (see the grouping parameter descriptions in Appendix A). Any detection that is not assigned to a teleseismic, local, or regional group is assigned a stassid of -1.

Phase Identification

The third major task in station processing is preliminary seismic phase identification. The current default implementation uses simple rules to identify teleseismic and local phases, and a more sophisticated approach based on Bayesian analysis to identify regional phases. We plan to use an approach similar to the one used in initial wave-type identification to enhance **StaPro** to allow station-specific rules to override and supplement this default method.

Teleseismic and Local Phases. The preliminary phase identification is determined first for teleseismic and local groups. For teleseismic groups, the first detection is identified as *P* and all subsequent detections are identified as *Tx* (unknown teleseismic detections). Isolated teleseismic detections (stassid = -1) are also identified as *P*. The *generator* of a local group will be identified as *Pg* and any other *P*-type detections in that group will be identified as *Px* (unknown *P*-type detection). The *S*-type detection in a local group will be identified as *Lg* unless its frequency is low and it is close to the *generator* (see the *min-lg-frequency* and *max-rg-time* parameters). In that case it will be identified as *Rg*. Noise detections are assigned a preliminary phase identification of *N*.

Regional Phases. Regional *generators* will be identified as *Pn* if there are no *S*-type detections in the group. Subsequent *P*-type detections in these groups will be identified as *Px*. Isolated *P*-type detections are identified as *Pn*, and isolated *S*-type detections are identified as *Sx* (unknown *S*-type detections). Regional groups with at least one *S*-type detection are more complicated. First, either the first *S*-type detection in the group or the largest one is identified by Bayesian analysis (see below). A similar analysis could be used for the *generator* (first *P*), but the current implementation just identifies it as *Pn* or *Pg* on the basis of *S-P* time. Once the *generator* and one *S*-type detection are identified, other detections are identified by prediction (see below). Finally, any remaining detections in the regional group are identified as *Px* or *Sx*.

A Bayesian analysis technique is used to identify either the first *S*-wave in a regional group or the largest one [*Bratt et al.*, 1991, 1994; *Bache et al.*, 1993] based on a static table of empirically-determined conditional probabilities (see the *bayes-file* parameter). The Bayesian analysis will estimate probabilities for all possible *S*-type regional phases (*Sn*, *Lg*, *Rg*, and *Sx*). The regional phase having the highest probability is the most-likely candidate, but additional constraints are also applied [*Bratt et al.*, 1991, 1994]. The conditional probabilities are based on velocity, horizontal-to-vertical power ratio, *S-P* time, period, and context with respect to other *S* phases. The context categories are: *only-S, first-S-of-two, first-S-of-many*, or *largest-S* (currently not used). The static table can include the conditional probabilities for multiple stations, and a default table can be specified for uncalibrated stations.

The prediction procedure begins by estimating a preliminary event location using the arrival time and azimuths of the *generator* and the *S*-type detection whose phase identification was determined by Bayesian analysis. The arrival time of one of the remaining regional phases (*Pg*, *Lg*, *Rg*) is then predicted. Three conditions must be met for a phase to be predicted: 1) the phase cannot already exist in the association group, 2) there must be an arrival in the association group with a compatible initial wave-type, and 3) the event depth and distance must be consistent with constraints for the phase being considered (see the *phase-distance-file* parameter). If an arrival is

found that is consistent with the prediction, then its phase is identified and it is used to refine the preliminary event location. The procedure continues until all valid regional phases have been considered.

Event Location and Magnitude

The final task in station processing is to compute a single-station event location and estimate local magnitude for association groups that satisfy a user-specified event confirmation criteria. These are used by **GAassoc** to screen detections from small local events.

Event confirmation is based on a weighted-count of defining observations (arrival time, azimuth, and slowness). It uses individual weights for each type of defining observation; separate weights are available for primary arrival times, secondary arrival times, array azimuths and slownesses, and three-component azimuths and slownesses. An association group passes the event confirmation test if its weighted-count exceeds a user-defined threshold (see the *event-confirmation-threshold* parameter). Event locations are determined for these groups, and the weighted-count test is applied again. This is done because the locator algorithm may reduce the number of defining attributes causing a reduction of the weighted-count. Finally, an additional confirmation test based on the size of the error ellipse is applied (see the *semi-major-axis-threshold* parameter).

Local magnitudes are estimated for confirmed events using the method described by *Bache et al.* [1991] in their Appendix A. Magnitude coefficients must be available for the regional phases involved (see the *mag-coefs-table* parameter). The local magnitude computed by **StaPro** is a weighted average of individual regional phase magnitudes. The phase magnitudes are based on the short-term-average amplitude measured on a 2-4 Hz incoherent beam (arrays) or vertical channel (three-component stations).

3.1.2 Data Flow

The data flow through **StaPro** is summarized in Figure 3. In this section, we describe how the input and output data are represented. Input data for **StaPro** come from parameter files, database relations, and static files. Separate sections are included below for each of these representations. Data are initially read into a global data structure. This structure is updated as results from the major modules are determined. **StaPro**'s final results are written to the RDBMS through our Generic Database Interface, or GDI [*Anderson et al.*, 1994]. Some output may also be written to ASCII log files if desired. Before describing each data representation, the **StaPro**'s process control is summarized

Process Control

StaPro has four input parameters that define the process control: station name, start time, end time, and duration (either duration or end time are specified, not both). During normal operations the Task Controller initiates **StaPro** for a given station and time interval, *t1* to *t2*. However, to accurately interpret data in this interval, **StaPro** must read data beyond it. Therefore, **StaPro** has been configured to automatically process data in overlapping windows to avoid the edge effects that would result from processing adjacent windows. The amount of overlap is determined from

six phase grouping parameters that control the maximum duration of a group of associations (see Appendix A). This overlap guarantees that **StaPro**'s final results for an interval will not depend on how that interval was segmented.

Figure 4 shows how **StaPro** sets up its processing intervals. The user (or Task Controller) specifies the interval to process as *t1* to *t2*. **StaPro** reads data from the RDBMS for an extended interval between *t1*-2Δ-*tnet* and *t2*+2Δ+*tnet*. This is called the query interval in Figure 4. The time, *tnet*, is added only to form contextual input parameters for the neural network used in initial wave-type identification [*Sereno and Patnaik*, 1993]. It is specified in the neural network weights table and is zero if the neural network is not used. However, all new **StaPro** processing is restricted to the maximum processing interval shown in Figure 4. The primary overlap interval, Δ, is determined from user-specified phase grouping parameters. It is defined as the maximum time interval between phases that belong to the same stassid group.

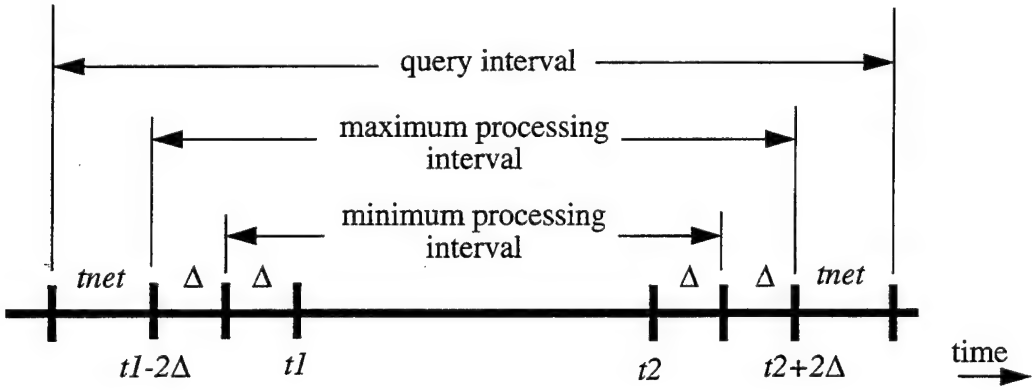


Figure 4. **StaPro** processes data from the RDBMS for the maximum interval shown above to avoid edge effects resulting from segmenting a long time window. The user specifies the interval from *t1* to *t2*, and **StaPro** automatically determines the overlap interval, Δ, from user-parameters.

StaPro immediately dissolves all previous station processing results for detections in the minimum processing interval shown in Figure 4. All of these detections will be reprocessed. It also dissolves previous results for detections in the surrounding intervals *t1*-2Δ to *t1*-Δ and *t2*+Δ to *t2*+2Δ that are part of a stassid group that was dissolved in the minimum processing interval. **StaPro** then processes the entire maximum processing interval, but skips detections if their stassid group was not dissolved.

Parameter file

A **StaPro** parameter file containing station-dependent processing parameters and general control parameters is required. It contains the control parameters described in the previous section, data-

base parameters, and numerous processing parameters which may be considered to be station-dependent (these are the *Par-File Parameters* in Figure 3). The **StaPro** manual page provides a complete description of all parameters and their default values (Appendix A). Station Processing can be customized by creating separate parameter files for each station.

Database

Standard CSS 3.0 database relations and their IMS Extensions contain the detections and their features needed by **StaPro** [*Anderson et al.*, 1990; *Swanger et al.*, 1993]. These include **arrival**, **apma**, **detection**, **amp3c**, **sbsnr**, and **mag_coefs**. **StaPro** also writes its output to standard CSS 3.0 database relations (Figure 3). It updates the **arrival** table, and writes new records to the **stassoc**, **assoc**, **origin**, and **origerr** tables.

The database information read by **StaPro** is assembled into a global data structure. Several database queries are required to fill this structure. A main query obtains arrival information by joining the **arrival**, **apma** and **detection** tables for the station requested. Next, horizontal-to-vertical power ratios are obtained from the **amp3c** table joined with the arrival table using similar constraints as the main query. This query is executed whether or not the initial wave-type neural network is used (see the *nnet-weights-file* parameter). If it is used and the horizontal-to-vertical power ratios are not available, then the neural network will exit with an error status causing the default rules to be used to identify the initial wave-type. The next query obtains the short-term average amplitude measurements from the **sbsnr** table for the user-specified channel (see the *sbsnr-chan* parameter). If these are not available, amplitudes from the **arrival** table will be used instead. The **arrival** and **sbsnr** tables are joined using similar constraints as the main query.

A separate structure is populated if the neural network is used to identify initial wave type (see the *nnet-weights-file* parameter). This structure contains contextual information required by the neural network for each arrival. The query reads the **arrival** table for all arrivals within the query interval shown in Figure 4.

Magnitude correction coefficients are read from the **mag_coefs** table for the station and channel specified. These coefficients are used in determining the local magnitude. Magnitude is not computed if the coefficients are not available.

Static files

Static input files used by **StaPro** include site information, neural network weights, default rules, optional station-specific rules (SSR), conditional probabilities for Bayesian analysis, allowable distance and depth ranges for seismic phases, and travel time tables. Optional output files are the neural network and locator log files. All of these are ASCII files.

The site file contains information about each station (e.g., location, station type). This information is also in the RDBMS, so we plan to eliminate the static site file in the next version of **StaPro**. Site information for the station being processed is stored in a global site structure. This file is currently required input for **StaPro**.

Three of the static files are specific to initial wave-type identification: neural network weights, default rules, and SSRs. Of these, only the default rules are required input for **StaPro** (although we recommend using the neural network for processing data from three-component stations). The default rules and SSRs are stored in CLIPS files. The default rules file (StaPro-IWT.clp) is included in the **StaPro** software release. This file should not be modified, but it provides valuable examples that can be used to develop SSR files. The neural network weights file can include weights for multiple stations.

Three of the static files are used for phase identification: the conditional probabilities, distance and depth restrictions for seismic phases, and travel time tables. The conditional probabilities are used in the Bayesian analysis to identify regional *S* phases. Tables for multiple stations can be stored in the same file. The distance and depth restrictions for seismic phases are used during phase prediction. The travel time tables are used for prediction and event location. The travel time tables are stored as separate files for each phase. They are required input for **StaPro**.

3.1.3 Major Software Components

This section lists the major software components of **StaPro** with a short description and the name of the file where they can be found. **StaPro** has been developed and tested on UNIX workstations under the Solaris 2.3 Operating System using Version 7.1.3. of Oracle™ and Version 6.0 of CLIPS. The major shared libraries used by **StaPro** are **libloc**, **libgdi**, **libdb30qa**, and **libpar**. The **StaPro** manual page is given in Appendix A.

main (int argc, char **argv)
main.c
Main() initializes CLIPS and the database, and it controls the flow of **StaPro** processing. A global parameter structure holds all user-parameters. A flag stating whether or not the neural network weights are available (e.g. wts_available) is set. This is needed for querying the database and for the CLIPS fact list. The processing flow consists of: querying the database, pre-processing, initial wave type, grouping, regional phase identification, event location and magnitude, and storing results in the database.

select_detection_data (double delta_t)
select_detection_data.c
This function builds a global data structure (data) by querying the **arrival**, **apma**, **detection**, **amp3c**, and **sbsnr** database tables. It also builds a time list data structure (tlist) if the neural network weights are available (wts_available) which is used to determine the contextual attributes for the neural network. The **mag_coefs** database table is queried and coefficients are stored in their own structure (mc). Finally, the static site file is read and stored in its structure (site).

iwt (void)
iwt.c
This function loops over each arrival. It asserts high-level facts, database facts, and user parameter facts onto the CLIPS fact list. Then, it begins firing the CLIPS rules for initial wave-type identification.

grouping (void)
grouping.c
This function identifies consistent groups of arrivals. All arrivals are examined for teleseismic groups. Phases (*P, Tx*) are assigned and stored in the global data structure (data) as the groups are formed. Similarly, local groups are found and assigned local phases (*Pg, Lg, Rg*). Lastly, regional groups are identified. However, regional phases are not assigned here unless the group is degenerate (i.e., does not contain any *S* phases). In this case, phase assignment is limited to *Pn* for the generator and *Px* for coda phases. Other regional groups will have their phases assigned during the Regional Phase Identification process. A stassid is assigned to all arrivals of the same group. The stassid is saved in the global data structure.

regional (void)
regional.c
This function determines phase types for regional groups. Bayesian analysis is used to determine the first or largest *S* phase. The *generator* is assigned phase *Pn*. A phase prediction procedure is used to identify other phases (*Pg, Lg, Rg*). Default phases (*Px, Sx*) are assigned to the remaining arrivals in a regional group. Phase identifications are stored in the global data structure (data).

locate (void)
locate.c
This function computes a single-station location and average local magnitude for events which pass the event confirmation criteria.

update_database (void)
update_database.c
This function writes **StaPro** conclusions to the database. Real values for stassid and orid are obtained from the database to replace the ones generated internally by **StaPro**. This is done prior to any database writes so that a clean database rollback can be achieved if a premature exit is required. The iphase, stassid, lddate and auth fields of the **arrival** table are updated from information stored in the global data structure (data). In addition, azimuth, delaz, slow, and delslo are updated in the **arrival** table for three-component stations. Database deletes are made from the **stassoc**, **assoc**, **origin**, and **origerr** tables. The dissolved stassid list is used to delete old stassids from the **stassoc** table. Similarly, the stassid list is used to delete old orids which might exist in the **assoc**, **origin**, and **origerr** tables. Finally, new tuples are inserted into the **stassoc**, **assoc**, **origin** and **origerr** tables. These tuples are built from information stored in the global data structure (data), and assoc, origin, and origerr structures.

3.2 GA Subsystem

The *GA Subsystem* includes two programs: **GAcons** and **GAassoc**. In addition, **libGA** is a library that includes subroutines and structures common to these two programs. The library provides the interface to a file containing grid information written by **GAcons** and used by **GAassoc**. The user may choose to divide the grid into several sectors and process them in parallel to accelerate the computations in **GAassoc**. **GAcons** builds one file for each sector specified by the user. **GAassoc** uses the grid information and the arrival data read from the database to form a list of preliminary events which is then passed to **ESAL** for further conflict resolution and event refinement.

The top-level view (Level 0) of the *GA Subsystem* was shown in Figure 2. This showed the interaction with the RDBMS and other elements of the *Global Association System.* The DFDs (Data Flow Diagrams) in this section decompose the Level 0 diagram into more detailed functional elements. The consistent notation used in these diagrams provides continuity between them. Figure 5 shows the Level 1 decomposition. **GAcons** and **GAassoc** are numbered 1 and 2. Later sections show the Level 2 decomposition of 1 (**GAcons**) and 2 (**GAassoc**) into functional elements 1.1, 1.2, etc. and 2.1, 2.2, etc. Level 3 decompositions are also provided. These require three digits to name each unit, and this convention allows the individual modules to be traced back to the previous level. At any level, the DFD provides a view of the system as an interaction between modules. The bubbles represent functional elements of the system at the specific level of decomposition. The solid arrows indicate data exchange between modules and data stores. The crosshatched arrows indicate data input by the user.)

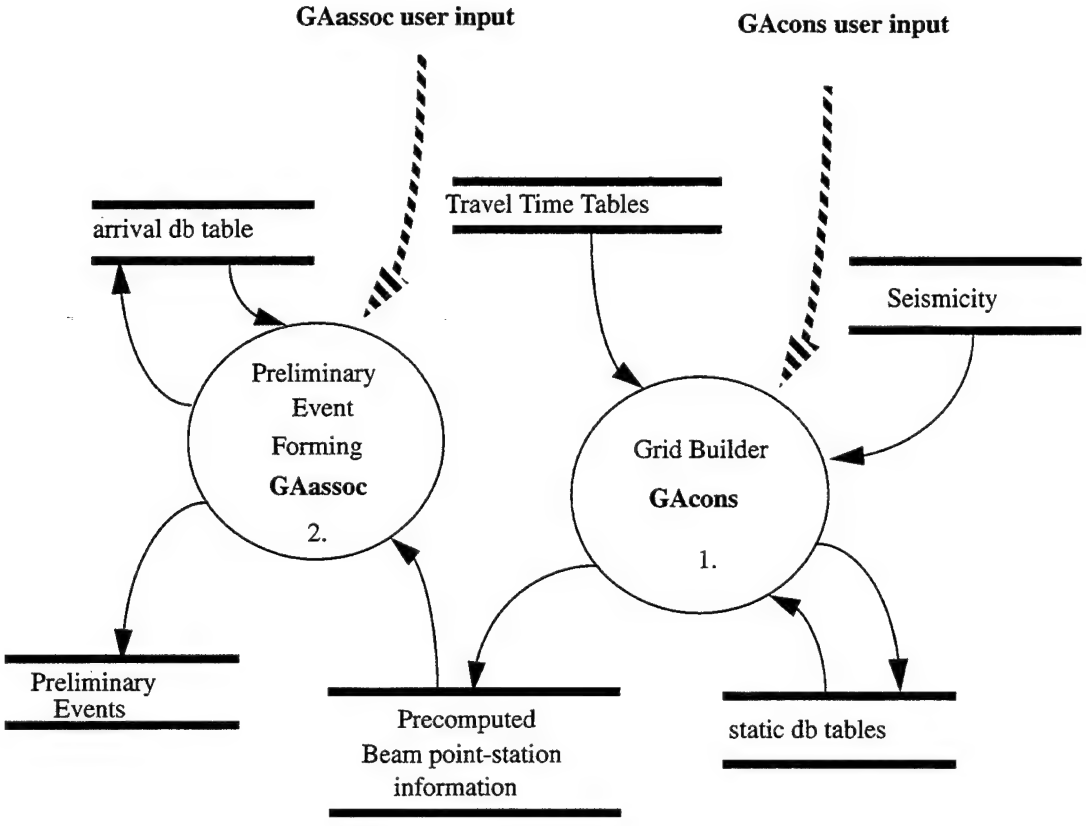


Figure 5. This is the Level 1 DFD for the *GA Subsystem.* The two functional units at this level are **GAcons** and **GAssoc.**

The following three subsections provide detailed explanations of the individual parts of the *GA Subsystem.* An explanation of the algorithms, data flow, and major software components are provided.

3.2.1 Grid Program (GAcons)

GAcons builds a global grid of precomputed information for use in the association program. The first task of **GAcons** is to establish a quasi-uniformly distributed set of points on a sphere. The points represent the centers of circular surface cells providing a complete overlapping coverage of the Earth. The grid is completed by adding depth cells with their center at the same latitude and longitude as the surface cells and their depth at user-specified values. The depth cells are added in areas where deep seismicity is known to occur.

For each cell, **GAcons** establishes a list of stations that have a non-negligible probability of detecting the earliest arrival for an event within the cell. The probability level is specified by the user (e.g., one chance in a hundred). Events are simulated for each grid cell. Their locations are distributed uniformly within the cell volume and their magnitudes are distributed according to a user-specified recurrence rate (i.e., *b*-value). A network probability subroutine (**probdet**) determines the probability of detecting these events at each station in the network based on attenuation models and noise estimates. For each simulated event, the station that records the earliest arrival is added to the list of "first-arrival stations."

Information about each grid cell-station pair is computed for all stations in the network. This includes information that pertains to all phases in the cell (e.g., epicentral distance) and information that pertains to a specific phase (e.g., travel time). A description of the precomputed information and the structure of the **GAcons** grid file is given in Section 3.2.3.

3.2.1.1 Algorithms

The algorithm used by **GAcons** to build the global grid is divided into three sections below: surface cells, depth cells, and "first-arrival stations." User-parameter referred to in this section are described in detail in Appendix B. We use the following distinct font and style to identify these parameters: *user-parameters*.

Surface cells

Grid points are established on the Earth with a quasi-uniform spacing. The grid points are distributed along parallels. The spacing in latitude is uniform and the number of latitude lines between 0 and 90 degrees computed as:

$$Nlat = \frac{\sqrt{2}}{\Delta} 90° \qquad (1)$$

where Δ is the average grid spacing provided by the user in degrees (see the *grid_spacing* parameter). The square-root of two provides a large enough density in latitude to ensure complete coverage. There will be a point at the North pole and the South pole.

The points are spaced uniformly for each latitude line. The number of points and the uniform spacing between them are computed using equations (2) and (3) where λ is the latitude. The grid is built by starting with uniformly distributed points on the equator with one point at longitude=0.

On the next latitude line, a point is placed at longitude, $\delta(0)/2$, exactly half-way in longitude between two points at the equator. This process is repeated for each latitude line, where one point is placed exactly half-way between two points at the latitude directly below. At lower latitudes, this results in a quasi-quincunx pattern rather than a square grid.

$$N(\lambda) = \mathbf{1} + \frac{(\cos\lambda)\,90^\circ}{\Delta} \tag{2}$$

$$\delta(\lambda) = \frac{180^\circ}{N(\lambda)} \tag{3}$$

The radius of each surface cell is set to the latitude spacing. This provides a sufficiently large overlap to ensure complete coverage of the Earth.

The current heuristic approximates uniformity. One improvement would be to optimize the set of grid points by minimizing the difference between segments joining points at a latitude and points at the previous latitude. An even more elaborate optimization would consist of solving the global problem at once rather than latitude by latitude. However, we believe that the final outcome of the association process is not heavily-dependent on the specific gridding scheme used.

Depth cells

A file containing events recorded over many years from a published global seismic bulletin (e.g., the Preliminary Determination of Epicenters) guides the establishment of grid cells at depth. The user provides a set of depths and widths for the depth cells (parameters ***depth_points*** and ***depth_widths***). Every surface location is included in the grid, and the depth cells below are included if they satisfy a user-specified threshold on the event density from the seismicity bulletin (***min_num_events_per_10sq_deg***). The cell shape is a portion of a cone truncated at the lower and upper depth of the cell. The file name that contains the seismicity information is specified through the user-parameter ***event_file***.

First-arrival stations

A list of stations having a non-negligible probability of detecting the earliest arrival in the network is established for each grid cell. This is done by Monte-Carlo simulation.

The number of events to be simulated for each grid cell is specified (***nevents***). Only events that are detected at a minimum number of stations (***nsta_def***) are included in the count. Events are randomly generated within grid cells for magnitudes between a minimum magnitude (***min_mag***) and a maximum magnitude (***max_mag***). A seismicity *b*-value drives the magnitude distribution of events between these bounds. The event locations are distributed uniformly within the grid cell volume. The *b*-value may be set on a cell by cell basis, but it is currently set to 1.0 for all cells.

For each simulated event, the probability of detection at each station is estimated using the event magnitude and location, a model of the attenuation, and the noise characteristics at the station. The station-specific noise level and signal-to-noise ratio threshold are read from the **siteaux** table in the RDBMS [*Swanger et al.*, 1993]. A random draw at every station based on the proba-

bility of detection simulates whether or not the station detected the event. If a station detects the earliest arrival for more than a user-specified percentage of the events (*percent*) then it is added to the list of "first-arrival stations."

3.2.1.2 Data Flow

The data flow diagrams describing the exchange of data between different modules within **GAcons** are shown in Figures 6 and 7. Figure 6 illustrates the two main components of **GAcons**, namely the Static Grid Builder and the Precomputed Table Builder (including first-station and travel time information for each phase). Figure 7 is a further decomposition of the precomputed grid information component. Major modules are shown on the diagrams next to each functional element.

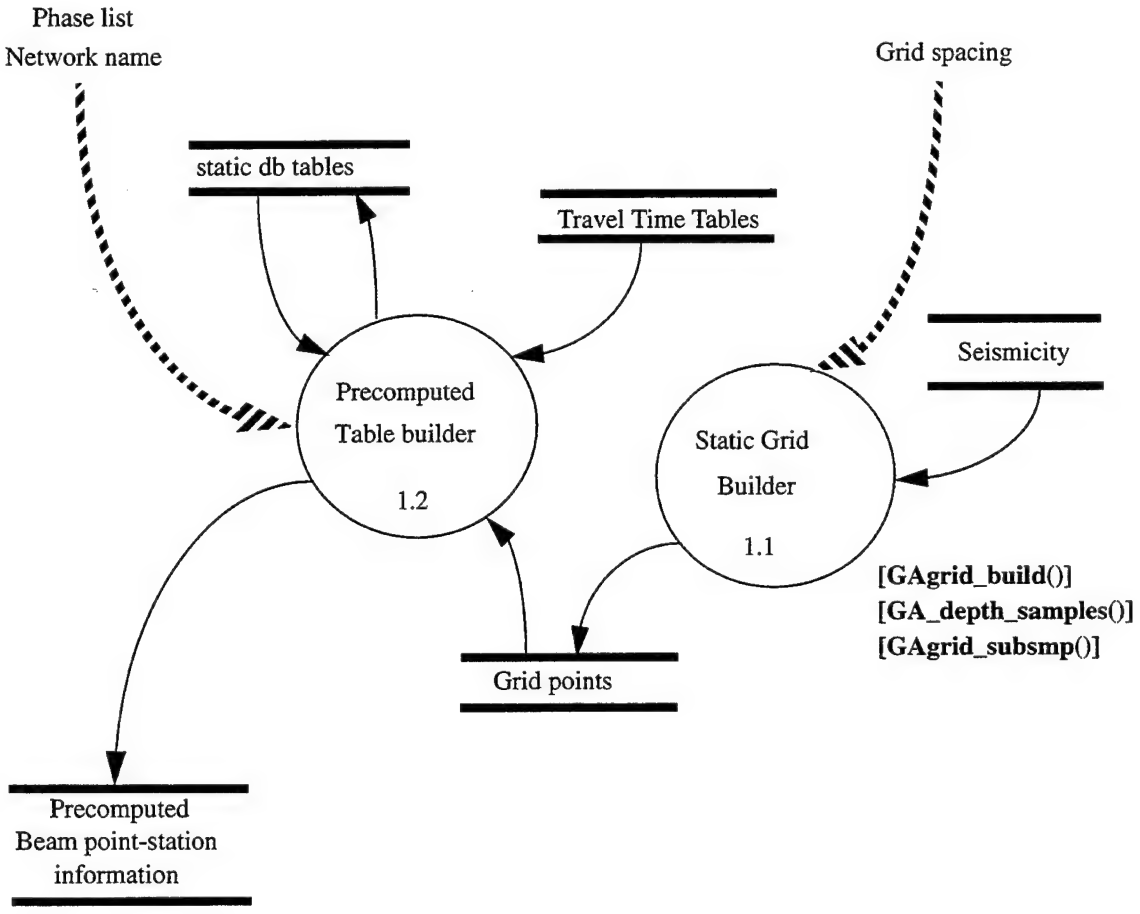


Figure 6. This shows the Level 1 DFD for **GAcons.**

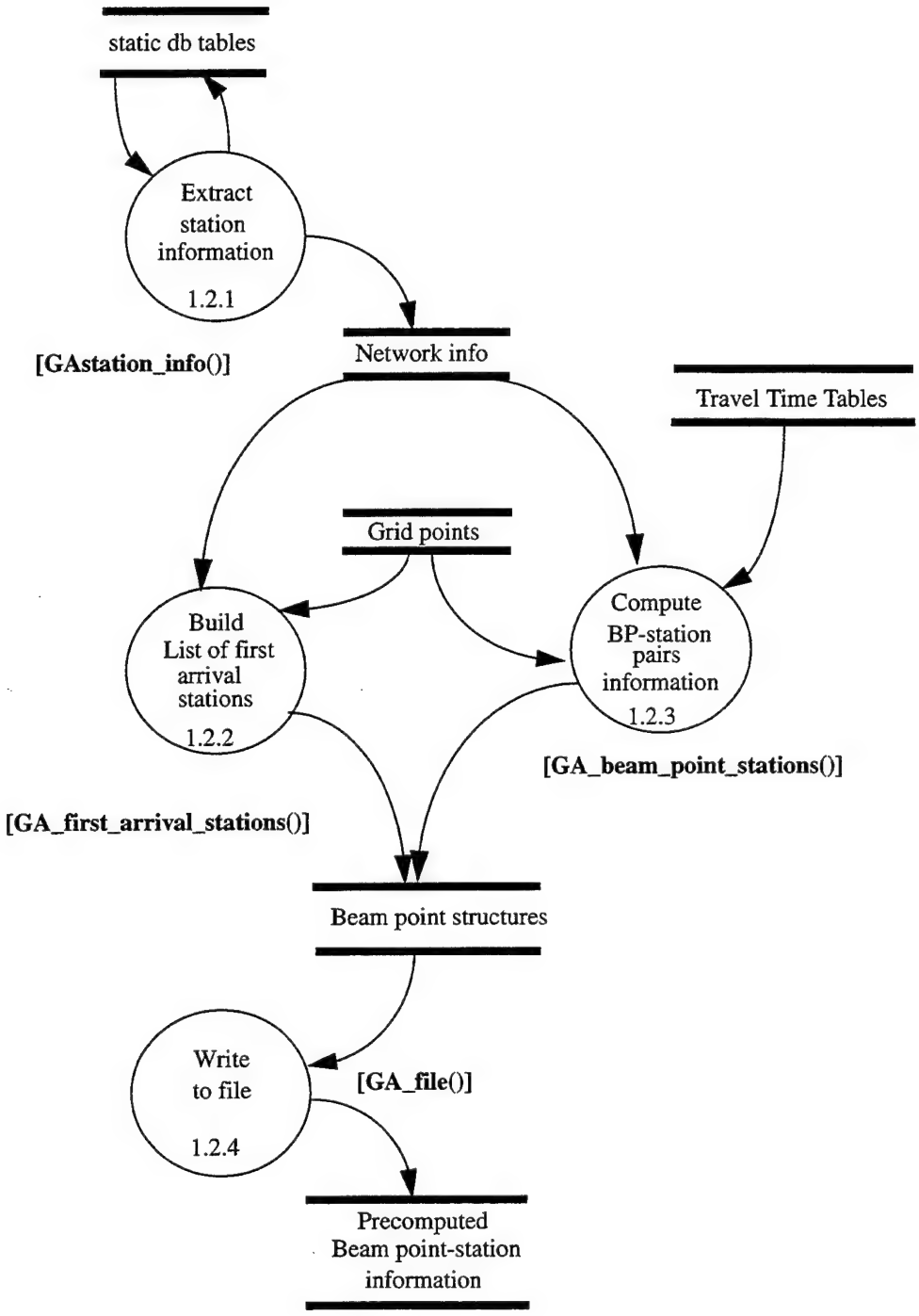


Figure 7. This is the Level 2 DFD for **GAcons**.

3.2.1.3 Major Software Components

This section presents tables with the standard ANSI C implementation of the major data structures, subroutines, and database tables used in **GAcons**. The data structures are described in Table 1and the relationships between them is shown in Figure 8. The major software components of **GAcons** are described in Table 2.

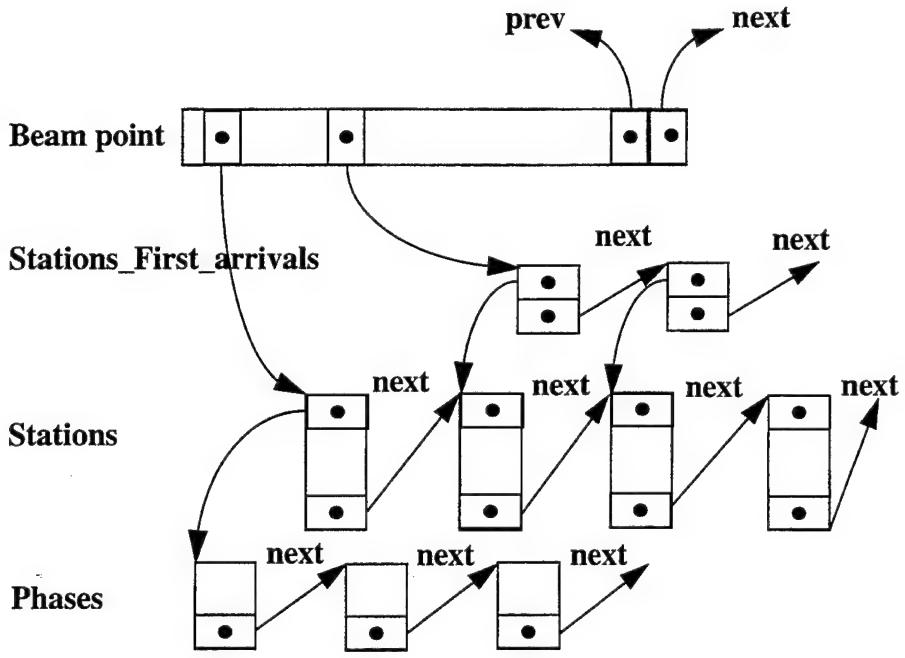


Figure 8. This diagram illustrates the data structures for grid cell-station information.

The Beam_pt and Grid_pt structures are instantiated once per grid point. Grid_pt contains the descriptive and geographical information for the grid points. Beam_pt links the Grid_pt structures to all the station information for that grid point.

The Sta_pt structure is instantiated once per grid point-station pair. A NULL-terminated linked list of Sta_pt structures is attached to the Beam_pt structure.

The Phas_Inf structure is instantiated once per grid point-station-phase triplet. A NULL-terminated linked list of Phas_Inf is attached to each Sta_pt.

The First_Sta structures contain a pointer to each "first-arrival station." They are organized in a linked list tied to each Beam_pt.

GAcons uses the Generic Database Interface (GDI) for all transactions with the RDBMS [*Anderson et al.*, 1994]. The database tables read by **GAcons** are **affiliation**, **site**, and **siteaux** [*Anderson et al.*, 1990; *Swanger et al.*, 1993]. These contain site-specific information for a given network.

Table 1: Major data structures in GAcons

List of the major data structures in **GAcons** with a short description and the name of the file where they reside.

Structure name and definition.	Short description of structures and data members.	File name
typedef struct grid_pt Grid_pt; **struct grid_pt {**		**libGA.h**
int index;	Unique grid point identifier number. For the southern hemisphere this is negative.	
float lat;	Latitude of current beam point.	
float lon;	Longitude of current beam point.	
float depth;	Depth of mid-point of cell.	
float lower_depth_bound;	Lower depth bound.	
float upper_depth_bound;	Upper depth bound.	
float radius ;	Radius of disk cell around grid point.	
float b_value ;	*b*-value in that cell.	
Grid_pt *next ;	Pointer to the next grid point. NULL if last.	
Grid_pt *prev ;	Pointer to the previous grid point. NULL if first.	
};		
typedef struct beam_pt **Beam_pt ;**	Beam point structure. Contains a pointer to the static grid point structure that it is associated with, plus pointers to the StaPt structures containing the Grid-point - Station information.	**libGA.h**
struct beam_pt **{**		
Grid_pt *loc ;	Pointer to Grid_pt structure containing location information.	
First_Sta *first_sta ;	Stations with potential for recording first arrival.Type First_Sta is a linked list of pointers to stations. The pointer will be specific to stations in the StaPt list.	
StaPt *stpt ;	Pointer to the linked list of all stations.	
Beam_pt *next ;	Pointer to next Beam_pt. NULL if last.	
Beam_pt *prev ;	Pointer to previous Beam_pt. NULL if first.	
} ;		
typedef struct first_sta **First_Sta ;**	Linked list of first arrival stations.	**libGA.h**
struct first_sta {		
StaPt *sta_pt ;	Pointer to station-beam point info.	
First_Sta *next ;	Pointer to next First_Sta structure.	
} ;		

Structure name and definition.	Short description of structures and data members.	File name
typedef struct stapt StaPt ;	Linked list of stations	**libGA.h**
struct stapt		
{		
char sta[] ;	Station code.	
int tab_index ;	Index into **arrival** table for use in **GAassoc**.	
float delta ;	Distance from beam point.	
float azi ;	Azimuth between beam cell center and station (angle measured at station).	
float baz ;	Back-azimuth (angle measured at beam cell center).	
float min_mag ;	Minimum detectable magnitude at station.	
float mag_cor ;	Magnitude correction at center of cell.	
float d_mag_cor_dr ;	Radial derivative of magnitude correction.	
float d_mag_cor_dz ;	Vertical derivative of magnitude correction.	
Phas_Inf *Ppt ;	Pointer to phase information.	
StaPt *next ;	Pointer to next station. NULL if this is the last station on the linked list.	
} ;		
typedef struct phas_inf Phas_Inf ;	Linked list of phase information structures.	**libGA.h**
struct phas_inf		
{		
char ph_id[] ;	Phase id. Seismological naming convention.	
Bool prim ;	True if phase is primary.False otherwise. Used in GAassoc	
float ttime ;	Travel time to center of cell.	
float ttime_min ;	Minimum travel time.	
float ttime_max ;	Maximum travel time.	
float d_ttime_dr ;	Radial travel-time derivative at cell center.	
float d_ttime_dz ;	Vertical travel-time derivative at cell center.	
float delcell ;	Cell width in slowness vector space computed from min and max slowness and azimuthal aperture (width of cell as seen from station).	
Phas_Inf *next ;	Pointer to next phase. NULL if last.	
} ;		

Table 2: Major modules in GAcons

List of the major software modules of **GAcons** with a short description of their function and the name of the file where they reside.

Subroutine Prototype.	Short description of function.	File name.
int main()	main for GAcons	**GAcons.c**
int GA_dist_depth_ranges (Dist_Depth_Range **dist_depth_range, char *range_file, char **phase_list, int num_phases)	Reads the distance/depth range information for each phase type from a user-specified file.	**GA_dist_depth_range.c**
int GA_first_arrival_stations (Beam_pt *bmpt, Netwrk *net, int nsta)	Generates a list of stations susceptible of recording the first arrival for a given beam point cell. Also estimates the minimum magnitude detectable at the station for an event within the beam point cell.	**GAcons.c**
int GA_beam_point_stations (Beam_pt *bmpt, Netwrk *net, int nsta, Dist_Depth_Range *dist_depth_range)	Generates the beam point - station information such as travel time bounds, slowness bounds, maximum detectable magnitude, etc.	**GAcons.c**
double GA_fkdist (float slow, float slow_min, float slow_max, float azi_del)	Computes the maximum possible distance in fk space between a slowness vector for an event anywhere in the cell and an event at the center of the cell as seen from a given station.	**GAcons.c**

3.2.2 Main Association Program (GAassoc)

GAassoc uses the grid data generated by **GAcons** to associate arrivals processed by **StaPro** and form event hypotheses. Each instance of **GAassoc** executes a loop over all grid cells included in a file given as user-input. Each grid cell is examined as a potential location for a seismic event. Grid files can be generated for multiple sectors, and **GAassoc** can be run separately (in parallel) for each sector.

The major tasks performed by **GAassoc** for each grid cell are:

- *Identify a DRIVER arrival*
- *Search for corroborating arrivals*
- *Apply preliminary event confirmation criteria*

This results in a set of preliminary event hypotheses for each grid cell. After all grid cells in the current sector have been processed, **GAassoc** performs the following additional tasks for each event hypothesis:

- *Split degenerate hypotheses*
- *Eliminate redundant event hypotheses*
- *Eliminate unlikely event hypothesis based on probability of detection*
- *Locate events and apply outlier analysis*
- *Apply event confirmation criteria*

Each of these tasks are described in the following section. User-parameters referred to in this section are described in detail in Appendix B.

3.2.2.1 Algorithms

The algorithms used by **GAassoc** to associate phases and form events are divided into eight sections corresponding to the tasks listed in the previous section.

Tasks for each grid cell

Identify a DRIVER arrival

The first task for each grid cell is to identify *DRIVER* arrivals. A *DRIVER* is an arrival at one of a limited set of stations in the network that could record the earliest arrival for an event in the given grid cell. This set of stations was determined by **GAcons** and is stored in the grid file. The maximum number of stations that will be considered for any grid cell can be specified by the user-parameter, *num_first_sta*. **GAassoc** processes arrivals in time-steps. To ensure that all corroborating arrivals of interest are available in the time-step, arrivals are loaded from the user-specified interval, *start_time - lookback* to *end_time*, but only arrivals with arrival times before *end_time - lookback* are considered as potential *DRIVERS*. The *DRIVER* has to have been identified as *P*, *Pn* or *Pg* by station processing (**StaPro**) and it must satisfy constraints on its slowness vec-

tor. Once a *DRIVER* is found, a hypothetical event origin time, magnitude and location are estimated.

The constraint on the slowness vector is illustrated in Figure 9. The length of the difference between the observed slowness vector and the predicted slowness vector must be less than a threshold. This threshold is the sum of the standard deviation of the slowness for the arrival (delslo), multiplied by a user-specified factor (σ) and the maximum difference between the slowness vector to the center of the cell and to any other point in the cell (Δ cell).

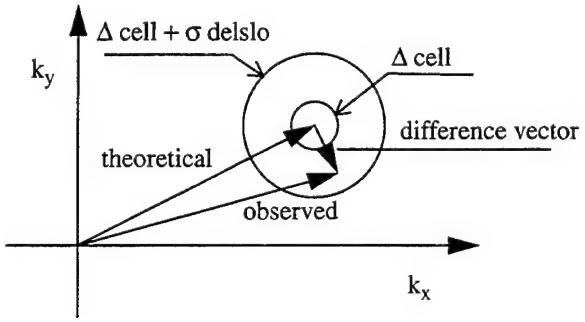


Figure 9. This illustrates the relationship between the theoretical and observed slowness vectors in wavenumber (*kx-ky*) space. The match between these two vectors depends on the length of the difference vector.

The standard deviation of the slowness for the arrival (delslo) is read from the database. User-parameter, *sigma_slowness*, specifies the factor σ. The maximum slowness vector difference (Δ cell) for the specific station, phase and cell is computed by **GAcons** and is stored in the grid file.

Search for corroborating arrivals

The next step in the association process is to search for arrivals that are consistent with the *DRIVER* (i.e., corroborating arrivals). This is done by examining arrivals at all stations at times later than the *DRIVER*, where the arrivals are restricted to phases in user-specified list (*phases*). Alternative phases to those determined by **StaPro** for these arrivals will be considered from a user-specified list (*forward_transformation_list*). If the confidence of the initial phase identification as read from the belief value in the **assoc** database table, is greater than a user-specified threshold (*belief_threshold*), then **GAassoc** will not override the phase identification determined by **StaPro**. An optional restriction can be imposed that a secondary phase may be associated only if there is an associated primary phase from the same station (*primary_required_for_secondary*). There is also a restriction that arrivals grouped by **StaPro** may not be associated if the group includes a regional *S* (specified by *regional_S_phases*) and α (*S- P* time) is less than the distance (in degrees) from the station to the center of grid cell (α is currently set to 0.1).

A simple screening process based on travel time and slowness vector (if available) is performed first. The theoretical arrival time is estimated from the observed arrival time, the travel time for the phase stored in the grid file, and an estimate of the event origin time derived from the *DRIVER*

assuming the location is at the center of the cell. The travel-time residual must be less than the sum of the time uncertainty (deltim), multiplied by a user-specified factor (*sigma_time*), and the maximum travel time across the grid cell. The maximum travel time across the cell is computed by **GAcons** and stored in the grid file. The slowness vector test is the same as that applied to the *DRIVER* described above.

A more rigorous chi-square statistical test is applied if an arrival successfully passes this initial screening. The chi-square test examines the compatibility between the corroborating arrival and the *DRIVER* arrival with an hypothesized event within the current grid cell. It uses all available features of the arrivals including travel time, slowness vector and amplitude. A location is performed using the time and slowness vector of both arrivals, when available (Figure 10). Assuming that the data and model parameters are normally-distributed, the location problem can be formulated as a least-squares problem which we solve by the QR decomposition method. The matrix elements for the least-squares location problem are stored in the grid file for each grid cell. They consist of the spatial derivatives of time, slowness vector components and magnitude. The model vector includes the origin time, magnitude (when amplitude data is available), and location (x, y and z) of the hypothesized event. The data vector is made up of the two travel time differences, the two magnitude residuals (when available), four components of the slowness vector (when available) and the x, y and z components of the location. The location coordinates in the data vector help stabilize the inversion. The model and data vector components are scaled by their standard deviations, so that the sum of squares of residuals can be interpreted as a chi-square sum which we compare to a user-specified threshold (*chi_limit*). The values for the standard deviations are important since they influence the chi-square sum. The values currently used for the standard deviations are: the radius of the grid cell for the x and y components of location, the half-depth of the cell for the z component, the estimated standard deviation (deltim) for the two time measurements, 0.3 magnitude units for the amplitude measurement and the magnitude, and the estimated standard deviation for the slowness vector components (delslo).

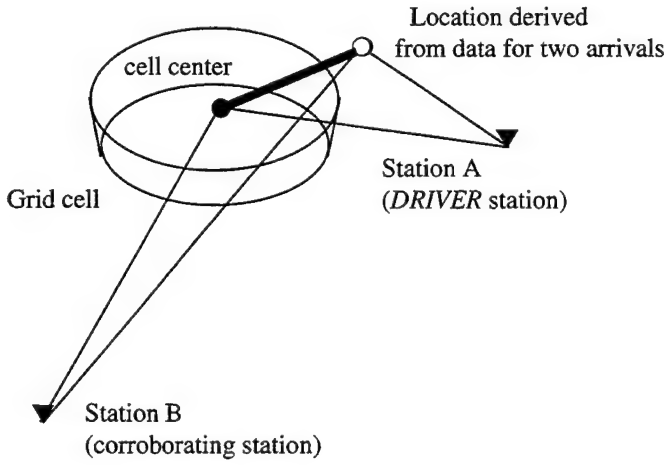


Figure 10. Schematic representation of the grid cell, its center, the stations corresponding to the *DRIVER* and corroborating arrival, and the location derived from these two arrivals.

As an efficiency heuristic, it is possible to freeze arrivals once they are associated with a preliminary event (*freeze_arrivals_at_beam_points*). Frozen arrivals will not be considered for association with any other preliminary event at the same grid point, but this will not affect their association with preliminary events at any other grid point.

Apply preliminary event confirmation criteria

When all possible corroborating arrivals have been examined for compatibility with the *DRIVER*, a weighted-count confirmation test is applied to the preliminary event. The following components are counted for the set of associated arrivals and multiplied by separate user-specified weights (*primary_time_weight, secondary_time_weight, array_azimuth_weight, 3comp_azimuth_weight, array_slow_weight, 3comp_slow_weight*):

- time for a primary phase
- time for a secondary phase
- azimuth at an array
- azimuth at a 3-component station
- slowness at an array
- slowness at a 3-component station

The event is removed from the list of preliminary events if the sum is less than a user-specified threshold (*weight_threshold*).

Tasks for each event hypothesis

Split degenerate hypotheses

The initial event construction can produce preliminary events that include incompatible arrivals such as two or more arrivals at the same station identified as the same phase, or the same arrival identified as two or more different phases. When this occurs, the degeneracy is split into two or more separate, self-consistent events. The event confirmation test is reapplied after the splitting since some of the resultant events may no longer pass the test.

Eliminate redundant event hypotheses

The same set of associations can be consistent with two adjacent grid points. It is also possible for a subset of associations to be consistent at an adjacent grid point. Consequently, a test can be applied after splitting to remove these redundancies from the list of preliminary events (*redundancy_required*). For an event to be deemed redundant, its arrivals must form a subset of another events's arrivals, and the arrivals in common must have been identified as the same phase. In addition, unless the *freeze_arrivals_at_beam_points* option is selected, the *DRIVER* must also be the same for both preliminary events. In the case of identical sets of associations, the one which best fits its grid cell is retained.

A double linkage is established when arrivals are associated with events. Each event includes pointers to its set of associated arrivals and, similarly, each arrival includes pointers to the events

with which it is associated. The algorithm used to check redundancy takes advantage this linkage, illustrated in Figure 11. To test whether an event (referred to as *Event A*) is redundant with another, its associated arrivals are examined and the arrival associated with the smallest number of events is identified. If that number is one, the event cannot be a subset of another. Otherwise, the other events connected to that arrival are examined and their arrival sets compared to that of *Event A*. If the arrival set of *Event A* is found to be a subset of one of these events then *Event A* is redundant. The benefit of using the double-linked structure is that it supports a reduced search space. The link from arrival to events allows the search to be restricted to events linked through at least one arrival.

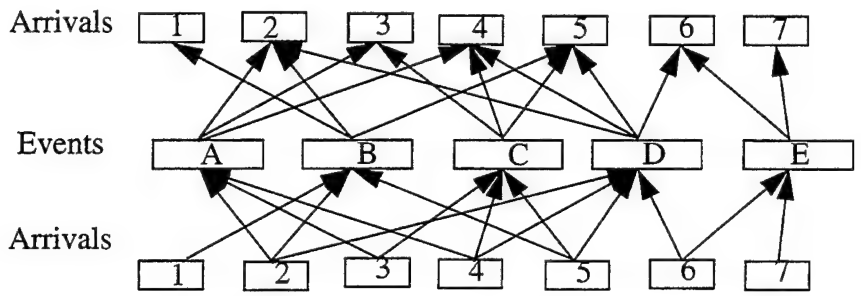


Figure 11. Linkage between preliminary events and arrivals. Arrivals are numbered from 1 to 7 and events from *A* to *E*. The same arrivals are repeated at the top and bottom of the figure. Events include pointers to their associated arrivals, and arrivals include pointers to events with which they are associated.

As an efficiency step, a partial redundancy analysis is performed during the association of corroborating arrivals if the number of preliminary events formed reaches a user-set limit (*count_limit*). This redundancy analysis is only applied to preliminary events built since the prior redundancy check.

Eliminate unlikely event hypothesis based on probability of detection

GAassoc has the option of applying a network probability test to preliminary events both before and after the event location is computed (*probdet_before_location*, *probdet_after_location*). The test is intended to screen unlikely small events that were not detected at a sufficient number of probable stations, and it is only applied to events where the number of associated primary phases is less than a user-specified limit (*max_obs_net_prob*). The advantage of applying it before location is that it can significantly reduce the number of event hypotheses that must be located.

The network probability test is based on determining for each preliminary event whether the set of stations at which primary phases have been detected is compatible with the location and magnitude of the preliminary event. The probability of detection at each station is computed from the estimated location and magnitude of the event, the location of the station, nominal values of the station noise, signal-to-noise detection threshold, and reliability (nois, noissd, snthrsh, and rely from the **siteaux** database relation), and an amplitude attenuation table (*atten_file*). When this

test is applied before location, the center of the grid cell is used as the estimated location and the estimated magnitude is the median of the individual station magnitudes. After location the computed location and magnitude are used.

The actual test takes the difference between the log of the product of the likelihoods for all station in the network (4) and its expected value (5) and normalizes this difference by the square root of the variance of the expected value (6).

$$\sum_{k,stations} \begin{cases} \log P_k & \text{if a primary phase is detected at the } kth \text{ station} \\ \log (1 - P_k) & \text{if a primary phase is not detected at the } kth \text{ station} \end{cases} \quad (4)$$

where k runs over active stations and P_k is the probability of detection at the *kth* station times the estimated reliability of the station

$$\sum_{k,stations} [P_k \log P_k + (1 - P_k) \log (1 - P_k)] \quad (5)$$

$$\left[\sum_{k,stations} P_k (1 - P_k) (\log P_k - \log (1 - P_k))^2 \right]^{1/2} \quad (6)$$

The hypothesis is eliminated if the expected value minus the actual value, divided by the square root of the variance, exceeds a user-specified threshold (***residual_over_sigma_max***).

Locate events and apply outlier analysis

Events remaining after this preliminary screening can be located and an analysis made of the residuals. Outliers are removed if necessary, redundancy checks performed, and the location refined. The locator uses the **libloc** library locator subroutine. Whether the locator is called or not is controlled by the parameter ***location_required***. Whether the location is computed at fixed or free depth is controlled by the parameter ***loc_fix_depth***. The fixed depth option uses the depth of the center of the grid cell. The location residuals are analyzed in terms of a chi-square test. Residuals for all data components are used simultaneously to compute a chi-square value. An outlier is defined to be an arrival whose chi-square value is larger than a user-set threshold (***chi_outlier***). The following procedure is then applied to each preliminary event to eliminate outliers:

Outlier analysis:
- If there are no outliers, proceed with event confirmation tests.
- If there are outliers.
  - Eliminate the worst outlier not including the *DRIVER*.
  - If the event is now redundant, remove it from the list.
  - Re-locate with the reduced set of arrivals.

Back to outlier analysis.

The redundancy test that is applied is the same as the one applied during association of corroborating phases except that the *DRIVER* is no longer required to be the same for both events.

Apply event confirmation criteria

A number of confirmation tests are applied to the event before it is written out to the database. These include the weighted-count test applied after association of corroborating phases, a supplemental restriction that the event have a user-specified minimum number of associated arrivals (*req_num_of_defining_detections*), and a restriction that the major axis of the error ellipse be less than a user-specified threshold (*max_smajax*). The same network probability test that was applied before location may also be applied at this point (*probdet_after_location*) using the computed event location and magnitude. Only events that pass all of the event confirmation tests are written to the database.

3.2.1.2 Data Flow

The data flow diagram describing the exchange of data between different modules within **GAassoc** is shown in Figure 12. Each submodule of **GAassoc** corresponds to a specific task. The subroutines used to accomplish each task are indicated in bold face and square brackets next to each module.

3.2.1.3 Major Software Components

This section presents tables with the standard ANSI C implementation of the major data structures and subroutines used in **GAassoc**. The data structures are described in Table 3 and the relationship between them is shown in Figure 13. The major software components of **GAassoc** are described in Table 4, and additional modules are described in Table 5.

The Arrival_Inf and Sta_Ar arrays of structures are initiated at the beginning of a **GAassoc** session from the **arrival** and **assoc** tables in the database. They are static during each execution of **GAassoc**. The Drivers structure contains preliminary events. These are built during the association loop. The Cor_Sta data structures contains information on the corroborating phases.

GAassoc uses the Generic Database Interface (GDI) for all transactions with the RDBMS [*Anderson et al.*, 1994]. The database tables are from the CSS 3.0 schema and its IMS Extensions [*Anderson et al.*, 1990; *Swanger et al.*, 1993].

3.2.3 Shared Libraries (libGA)

Modules common to **GAcons** and **GAassoc** have been grouped under the **libGA** library. This library includes the subroutines associated with building the grid, dividing it between sectors, and reading and writing the grid file. This section provides a detailed description of the file structure currently used.

3.2.3.1 Data Description

The grid file is a flat file containing the information pertaining to all beam points in a sector for the complete network of stations. The information is written sequentially by Beam Point Record. As

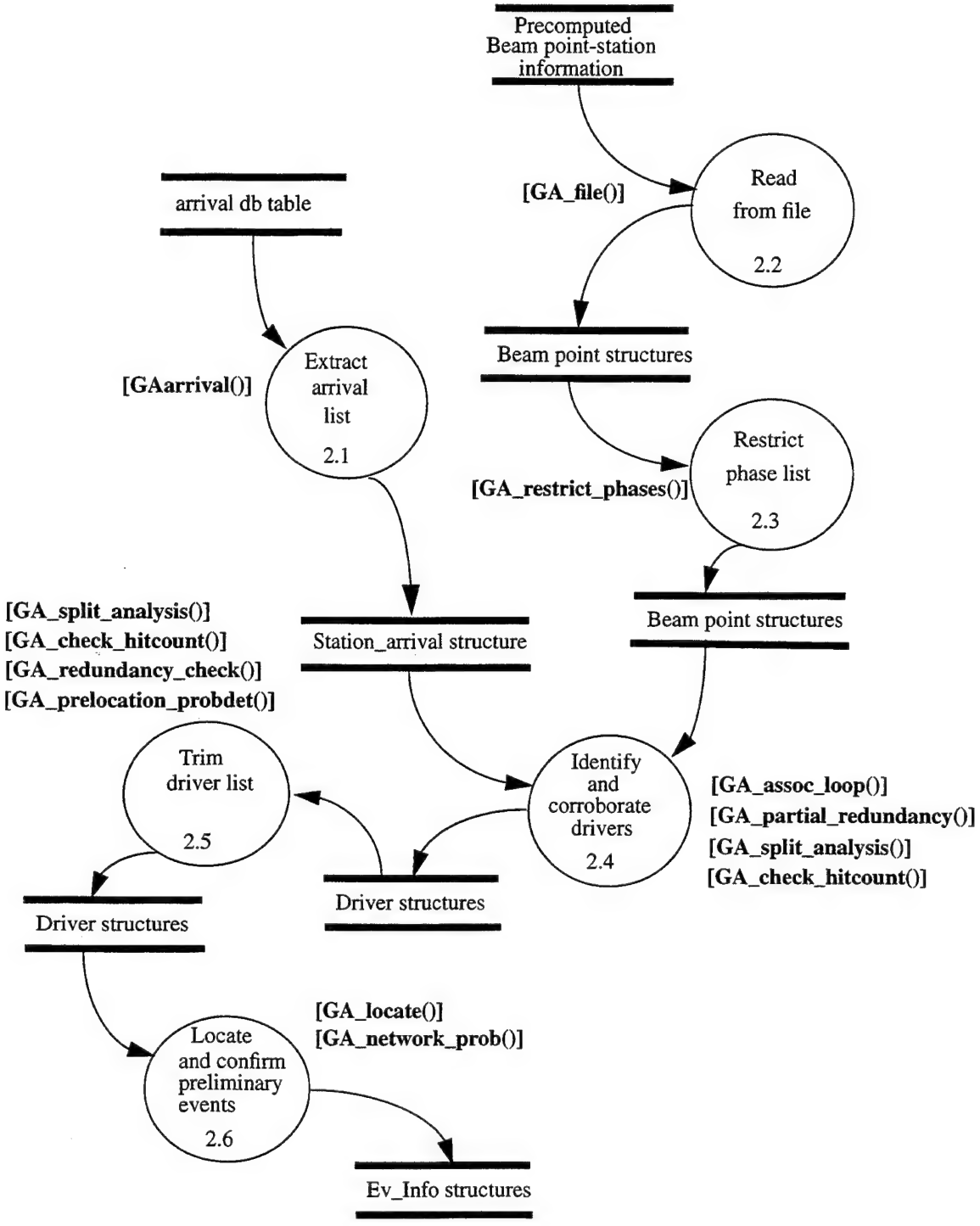


Figure 12. This is the Level 1 DFD for **GAassoc**.

Table 3: Major data structures in GAassoc

List of the major data structures in **GAassoc** with a short description and the name of the file where they reside.

Structure name and definition.	Short description of elements.	File name
typedef struct driver Driver;	Driver or generator. This is a structure formed to contain preliminary event information. It contains pointers to static station and arrival information and to the beam point where it has been formed.	**GAassoc.h**
{		
struct Beam_pt *bp;	Pointer to Beam-Point structure.	
char ph_id[ ];	Phase ID for driver arrival.	
StaPt *stpt;	Pointer to station information for beam point.	
Phas_Inf *phspt;	Pointer to phase information for this driver-phase-beam point.	
Sta_Ar *sta;	Pointer to station arrival structure for first arrival station	
Arrival_Inf *ar;	Pointer to Arrival_Inf structure for first arrival for this driver	
Cor_Sta *csta;	Pointer to corroborating stations and arrivals list	
double or_time;	Origin time for the driver	
double or_tmin;	Min. origin time for the driver	
double or_tmax;	Max. origin time for the driver	
double dr_mag;	Magnitude computed from this driver	
double qfact;	Quality factor. This is the combined probability for all corroborating phases to be associated with this driver	
double weight ;	Current 'weight' of driver obtained by adding all weights for associated arrivals	
int num_obs;	Total number of arrivals for this driver, including driver	
Driver *next;	Pointer to next driver.	
};		
typedef struct cor_sta Cor_Sta;	Corroborating station. Used to associate driver to corroborating arrivals.	**GAassoc.h**
struct cor_sta		
{		
Sta_Ar *sta;		
Arrival_Inf *ar;	Pointer to station detections info.	
double qual;	Pointer to arrival	
double mag;	Quality factor of the chi2 association	
char ph_id [];	Magnitude of corroborating association	
Cor_Sta *next;	Phase id for this arrival.	
};	Pointer to next Cor_Sta. NULL if last.	

Structure name and definition.	Short description of structures and data members.	File name
typedef struct arrival_inf Arrival_Inf;	Arrival information extracted from arrival table	**GAassoc.h**
struct arrival_inf		
{		
long arid;	arid for this arrival.	
double time;	time of arrival.	
char iphase[];	reported phase.	
double deltim;	uncertainty on measurement of time.	
double azimuth;	measured azimuth.	
double delaz;	uncertainty in azimuth.	
double slow;	measured slowness.	
double delslo;	uncertainty in slowness.	
double amp;	measured amplitude.	
double delamp;	uncertainty on amplitude measurement.	
double per;	measured period.	
double logat;	measured log of amplitude over period.	
double weight;	weight of observation based on user-defined coefficients. Each of the time, azimuth and slowness have a coefficient.	
double belief;	Belief field from assoc table if available.	
Stassid_pt *staspt;	Pointer to stassid structure if available. This will be NULL if no valid stassid available.	
int assoc;	Flag. assoc =0 if the arrival is not associated.	
int drl_count;	Driver count of drivers(events) associated with this arrival (detection).	
Dr_list *drl_anchor;	Pointer to the first driver in the linked list of drivers this arrival belongs to.	
Dr_list *drl_current;	Pointer to the last (current) driver in the linked list of drivers this arrival belongs to.	
};		
typedef struct ev_info Ev_Info;	Preliminary event structure built by the locator module.	**GAassoc.h**
struct ev_info		
{		
Origin origin;	Origin record	
Origerr origerr;	Origerr record.	
Arrival *arrival;	Pointer to arrival record.	
Assoc *assoc;	Pointer to assoc record	
int num_obs;	Number of observations associated to the event.	
int good_event;	Flag	
int loc_err_code;	Flag. Locator error code.	
Ev_Info *next;	Pointer to the next Ev_Info structure.	
};		

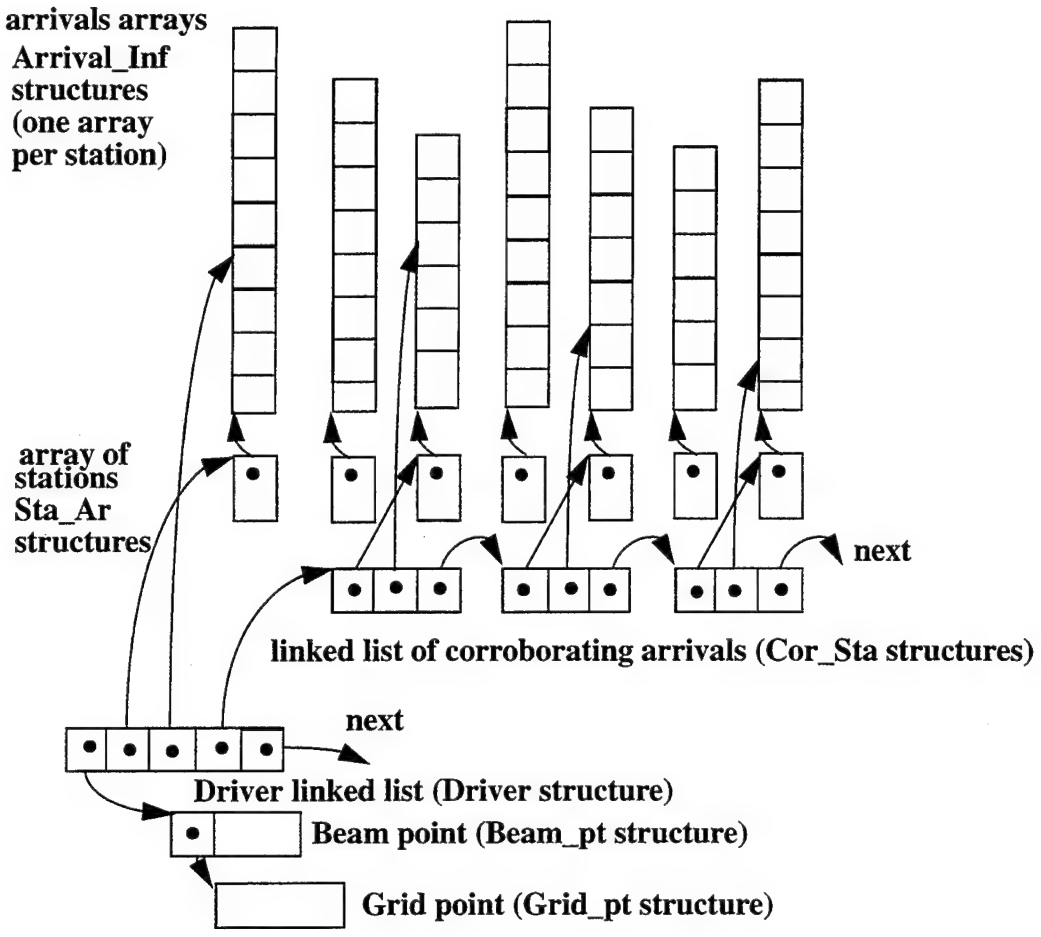


Figure 13. This diagram illustrates the data structures for **GAassoc**.

Table 4: Major modules in GAassoc

List of the major software modules of **GAassoc** with a short description of their function and the name of the file where they reside.

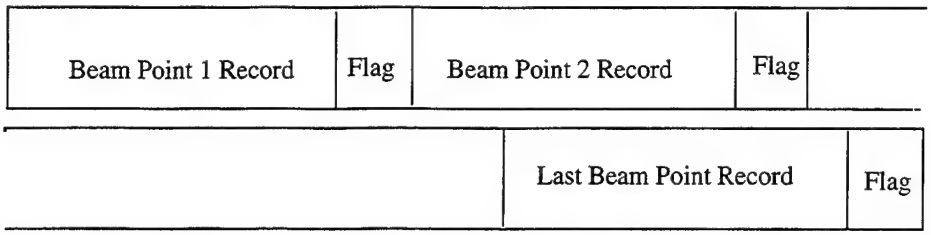
Subroutine Prototype.	Short description of function.	File name
int main()	main for GAassoc	**GA_assoc.c**
int GA_assoc_loop (char *filename, Driver **anch, Assoc_loop_params *params)	This subroutine performs the main task of the association program. It builds preliminary events and returns them in a linked list with anchor element anch.	**GA_assoc_loop.c**
void GA_partial_redundancy (Driver **driver, Driver **end, Driver **prev, Bool freeze)	Performs a redundancy check on part of the linked list of preliminary events (Driver structures). This is used during the association loop to reduce the number of preliminary events formed if this number reaches a user-set limit.	**GA_assoc_loop.c**
void GA_redundancy_check (Driver **driver, Driver **dr_prev, Bool freeze)	Performs a redundancy check on the complete linked list of preliminary events. This is called after the association loop.	**GA_assoc_loop.c**
void GA_split_analysis (Driver *driver)	Performs the split analysis on the whole linked list of preliminary events.	**GA_assoc_loop.c**
double GA_chi2_test (Driver *driver, StaPt *stpt, Phas_Inf *phspt, Sta_Ar *sta, Arrival_Inf *ar, int *free_deg, double *res_vec)	Performs the chi-square test analysis for a driver arrival and one of its corroborating arrivals. The corroborating arrival has been screened previously for arrival time and slowness vector. A QR method is used to solve the least-squares problem.	**GA_chi2_test.c**
void GA_check_hitcount (Driver **dr_anch, double threshold)	Checks the weights of all preliminary events in the linked list. Eliminate preliminary events which weigh less than a threshold value.	**GA_reduce.c**
int GA_locate (Ev_Info **ev_info, Driver **dr_anch, Locator_params *locator_params, Ev_Confirm *ev_confirm, double chi_outlier, Site *sites, int num_sites)	Locator module.This module locates the preliminary events and refines the events by doing an outlier analysis on them. The linked list of driver structure preliminary events is input into the module and a linked list of located and confirmed events (Ev_Info structures) is formed.	**GA_locate.c**

Table 5: Additional modules in GAassoc

These additional subroutines are called by the major components of **GAassoc**.

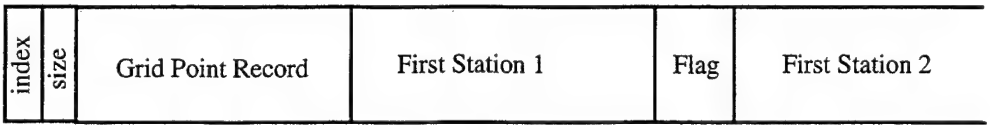
Subroutine prototypes.	Short description.	File name.
int GAarrivals (dbObj *arr_obj, Sta_Ar **ar_table, int *nsta, Ev_Confirm *ev_confirm, Site *sites, int nsites)	Interface subroutine. Builds an array of Sta_Ar elements and fills in the information from the dbobj arr_obj.	**GA_assoc.c**
int GA_destroy_driver (Driver **dr)	Releases memory previously allocated to element type driver and all associated corroborating arrivals.	**GA_assoc.c**
int GA_restrict_phases (Beam_pt *bmpt, char **phases, int nphases, char **primary_phases, int nprim)	Restricts the list of phases to be incorporated into the linked list for each beam point-station pair based on a user provided list.	**GA_assoc.c**
Bool GA_redundant_subset (Driver *driver, Bool freeze)	Determines whether preliminary event driver is redundant with another event within the linked list of preliminary events.	**GA_reduce.c**
int GA_same_arrival_count (Driver *driver)	Counts the occurrences of the same arrival within the preliminary event.	**GA_reduce.c**
int GA_same_phase_id_count (Driver *driver)	Counts the occurrences of the same phase id within the preliminary event.	**GA_reduce.c**
void GA_split_phase_id (Driver *driver, Cor_Sta *csta, Cor_Sta *cur)	Splits the preliminary event driver into two events. Corroborating elements pointed at by csta and cur shared the same phase id.	**GA_reduce.c**
void GA_split_arrival Driver *driver, Cor_Sta *csta, Cor_Sta *cur)	Splits the preliminary event driver into two events. Corroborating elements pointed at by csta and cur shared the same arrival.	**GA_reduce.c**
Driver * GA_split_driver (Driver *driver)	Calls either GA_split_arrival or GA_split_phase_id after GA_same_arrival_count or GA_same_phase_id count has determined redundancy of arrival or phase id.	**GA_reduce.c**

shown below, the structure of the file allows for a flexible number of beam points, stations and phases.

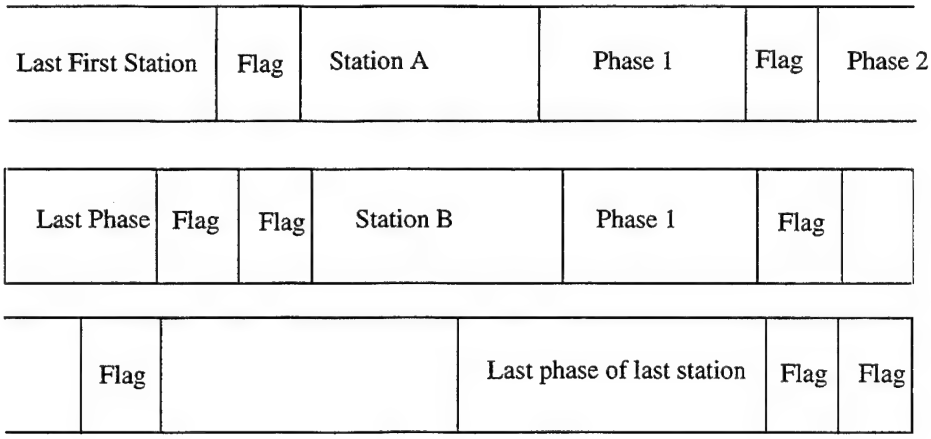


At the end of each Beam Point Record, a four-byte integer flag indicates whether or not there are more beam points. A value of one indicates that there are more, and a value of zero indicates the end of the file.

The first two elements of the Beam Point Record are the index number of the grid point and the size of the record. Following this header is a Grid Point Record containing static information such as latitude, longitude, radius of the cell, etc. The length of this record is equal to the size of the grid_pt structure (Grid_pt) as listed in Table 1. The next records contain the "first-arrival station" information. For each station that could detect the earliest arrival, a First Station Record of length equal to the size of the StaPt type is written (see Table 1). First Station Records are separated by a flag indicating whether or not another record follows the current one. This is shown in the diagram below.



Following the last First Station Record are structures containing Station Records for all stations in the network and Phase Records for all phases requested by the user. The Station Record comes first and it is followed by the first Phase Record (the length of this record is equal to the size of the Phas_Inf C type). All Phase Records are written in sequence separated by flags. The last Phase Record for a station is followed by a flag indicating there are no more Phase Records to follow. Two zero flags (one after the other) indicate the end of the last station for this beam point. This is illustrated below:



The Grid Point Record has the following format:

index	lat	lon	depth	lower_ depth_ bound	upper_ depth_ bound	radius	b_value	IGNORE	
int	float	float	float	float	float	float	float	pt	pt

The last two elements can be ignored when reading the file. They contain the pointers to the previous and next Grid Point Records which are unusable when reading from the file. This record matches the Grid_pt C type defined in Table 1.

Each Station Record contains the attributes characterizing the station-beam point pair. The format is shown below:

station code	delta	azi	azi_min	azi_max	min_mag	mag_cor	d_mag_ cor_dr	d_mag_ cor_dz	IGNORE	
char 6	float	float	float	float	float	float	float	float	pt	pt

A number of Phase Records are attached to each Station Record. These characterize the propagation between the station and beam point. All phases that can exist at the station for an event within the grid cell that have been specified by the user are included as Phase Records. The format of the Phase Record is shown below:

phase ID	prim	ttime	ttime_min	ttime_min	d_ttime_dr	d_ttime_dz	delcell	IGNORE
char 9	Bool	float	float	float	float	float	float	pt

3.2.3.2 Major Software Components

The major software components of **libGA** are described in Table 6, and additional modules are described in Table 7.

Table 6: Major modules in libGA

LibGA contains subroutines that are common to **GAcons** and **GAassoc**. The following subroutines are the major components of the library. A short description is attached to each subroutine and the file they reside in is listed.

Subroutine	Short description	File name
int GAgrid_build (double grid_spacing, Grid_pt **anchor)	Builds a quasi-uniform surface grid given a grid spacing in degrees and returns a handle to the first element in a linked list of Grid_pt structures.	**GAgrid.c**
int GAgrid_subsmp (Beam_spec reg, Grid_pt *gpt, Grid_pt **gsub)	Subsamples the whole earth grid given a region specified in Beam_spec reg, the pointer to the anchor element of the global grid (gpt), and returns the pointer to the subgrid (gsub).	**GAgrid.c**
int GAdepth_build (Depth_Grid *depth_grid , int Num_depth_bnd, Grid_pt *bmp, char *event_file, double Min_num_events_per_10sq_deg)	Adds the depth elements to the grid given the seismicity information and the parameters of the depth cells.	**GAgrid.c**
int GA_file (Beam_pt **anch, char *filename, int rw)	Writes or reads file containing grid information.	**GAgrid.c**

Table 7: Additional modules in libGA

The following list of subroutines are called by the major modules in **libGA**.

Subroutine	Short description	File
int GA_station_info (dbObj *sta_obj, Netwrk **net, int *nsta)	Transfers the station information from a dbobj into the Netwrk structure.	**GA_station_info.c**
void * GAgrid_destroy (Grid_pt **gpoint)	Releases memory allocated to the grid point linked list whose anchor element is passed as argument.	**GAgrid.c**
int GA_destroy_sta_info (StaPt **station)	Adds the depth elements to the grid given the seismicity information and the parameters of the depth cells.	**GA_station_info.c**
int GA_destroy_first_-sta_info (First_Sta **first_station)	Releases memory allocated to all first arrival stations in the linked list with anchor element first_station.	**GA_station_info.c**
int GA_cmpsta (const void *sta_inf1, const void *sta_inf2	Compares two Sta_Inf structures. This is for use in the general sorting function qsort. The sorting is done according to the grid cell to station distance.	**GA_station_info.c**

3.3 Expert System for Association and Location (EServer/ESAL)

The event hypotheses produced by **GAassoc** are processed by **ESAL** to resolve conflicts and to refine the hypotheses. **GAassoc** produces alternative event hypotheses which contain many conflicting associations of the same arrival. The hypotheses do not include any associations of late-arriving secondary phases and occasionally are missing associations of consistent primary phases. **ESAL** reads the event hypotheses produced by **GAassoc**, resolves the conflicts, and completes the set of associations. Specifically, **ESAL** performs the following functions to complete the **GAassoc** hypotheses:

- *Resolves conflicts when a single arrival is associated with multiple event hypotheses.*
- *Associates late-arriving secondary phases.*
- *Associates primary phases that may have been missed by* ***GAassoc****.*
- *Resolves edge-effect conflicts which can arise when* ***GAassoc*** *is parallelized to run separate processes on different sectors of the Earth.*

ESAL is a full-functioned knowledge-based automatic association program capable of processing arrivals from a single station (Station Processing) and from a network of stations (Network Processing). It is a mature program which has been used for a number of years in different systems with varying processing requirements. It has been used in IMS to process continuous data from a small regional network [*Bache et al.*, 1993], in ADSN to process continuous teleseismic data, and in the GSETT-2 experiment to process mixed regional and teleseismic data from a global network. It is programmed in the ART expert system shell (a product from Inference Corporation) and Common Lisp, with integrated C and Fortran libraries which perform earth-model calculations. **ESAL** has been described in detail elsewhere [*Bratt et al.*, 1991, 1994] and we will focus here on those features which have been added to allow processing of GA preliminary hypotheses[1].

The basic model used by **ESAL** for Network Processing is to generate a series of trial origin hypotheses using any of a number of different techniques (Trial Origin Methods) and to then iteratively develop the hypotheses by adding new associations, relocating, and removing inconsistent associations (the Association Loop). In this system, all new trial origins are generated from **GAassoc** hypotheses. Only one other trial origin method is used: Previous, which is used to complete events partially formed during the preceding time-step. The primary modification of **ESAL** for this system was the addition of a new Trial Origin Method which reads the **GAassoc** preliminary hypotheses and resolves the conflicts before passing them to the Association Loop.

1.**ESAL** is a highly parameterized program which allows considerable customization of its application to different networks and processing requirements. There are over 300 user adjustable parameters which are described elsewhere [*Bratt et al.* 1990, 1994]. In the following discussion we will reference only selected parameters of specific interest to the processing of **GAassoc** hypotheses. Descriptions of new parameters are given in Appendix C.

ESAL reads most of its input data from, and writes its output to, flat files in the external format of the CSS 3.0 database schema [*Anderson et al.*, 1990; *Swanger et al.*, 1993]. **EServer** provides the interface which moves data between these files and the RDBMS. **EServer** was modified for this application to write files corresponding to the **GAassoc** solutions for input to **ESAL**[1].

3.3.1 Conflict Resolution

The main task for **ESAL** in this system is the resolution of conflicts between different **GAassoc** hypotheses that associate the same arrival. **ESAL** has a conflict resolution mechanism that is used to resolve conflicts that arise in the course of normal **ESAL** processing, but this proved inadequate for two reasons: 1) It assumes that conflicts are resolved as they arise and that consequently there will never be more than a couple of origins in conflict at one time; this allows a conservative approach which becomes computationally very expensive as the number of origins in the conflict set increases, and 2) The resolution is not performed until the origin is completely formed (i.e., has passed through the entire Association Loop). **GAassoc** generally produces one to two orders of magnitude more hypotheses than the number of final events. The cost of passing all of the hypotheses through the Association Loop plus resolving the large conflict sets was found to be prohibitive.

Consequently, we implemented a new trial origin method, called GA, wherein the conflicts are resolved before any hypotheses are passed to the Association Loop. The preliminary hypotheses generated by **GAassoc** are read from a set of files in the CSS 3.0 external format of the **origin**, **origerr**, and **assoc** relations produced by **EServer**[2]. They are read into data structures parallel to, but distinct from, those used in the rest of **ESAL** processing. This is because **ESAL** is a rule-based system which makes extensive use of pattern matching on the data objects representing event hypotheses. It is very inefficient to perform all of the initial pattern matching on the large number of preliminary hypotheses that will not survive conflict resolution.

All conflicts with multiply-associated arrivals are resolved to produce a set of distinct hypotheses which can be used as trial origins. The conflicts are resolved on an arrival basis. Each arrival which is associated to more than one hypothesis is examined in the context of the hypotheses and is disassociated from all but the "best". Four different tests are currently available to determine which hypothesis is best[3], and the implementation was structured to make it convenient to add and evaluate alternative tests. Three of the tests evaluate the hypotheses exactly as they are input. These are: 1) smallest error-ellipse area, 2) largest number of defining phases[4]; ties are broken by smallest error-ellipse area, and 3) a composite test which is the same one **ESAL** uses elsewhere to resolve conflicts; this is based on an ordered list of criteria including: defining vs. non-defining, firmly-associated vs. weakly-associated, number of defining components, and error-ellipse area.

1.**EServer** parameters added to allow writing of files containing the **GAassoc** solutions include: *write_ga_-files*, *ga_origin*, *ga_origerr*, *ga_assoc*, and *ga_arrival*.

2.[*ga-origin-filename, ga-origerr-filename, ga-assoc-filename*]

3.[*ga-conflict-resolution-heuristic*]

4.A "defining" phase is one that has been used to the calculate the location of the hypothesis. A defining component is a phase attribute, time, azimuth or slowness, which has been used to the calculate the location of the hypothesis.

The fourth test is also based on the largest number of defining phases, but starts at the best hypothesis and progressively reduces the number of defining phases of each hypothesis that loses an association. This type of test, where the measure of a hypothesis is revised when it loses an association, is clearly desirable but not practical for many tests. Tests that depend on the error ellipse or residuals or any feature that requires re-calculation of the location would be very expensive given the number of conflicts that **GAassoc** typically produces.

When conflict resolution is complete, all remaining distinct hypotheses are checked to determine if they still satisfy a minimality condition; hypotheses that do not are abandoned. The check used is **ESAL**'s weighted-count event confirmation test. The number of defining time, azimuth and slowness components from arrays and 3-component stations are counted separately, multiplied by parameterized weights, and the sum compared to a threshold[1]. This is the same test that is used during **GAassoc** and later by **ESAL** during event confirmation.

Distinct hypotheses which pass the weighted-count test are used one at a time as trial origins, and are ordered by either origin time, number of defining phases, or body-wave magnitude[2].

In addition to the conflicts discussed above, which exist between preliminary hypotheses produced by a single **GAassoc** run, there can be conflicts due to edge effects produced by dividing the Earth into sectors and running **GAassoc** separately on each sector. These conflicts can be resolved either by processing them through **ESAL** all at once, or by running **ESAL** on the results of each **GAassoc** run and then merging the results in a subsequent run using **ESAL**'s normal conflict resolution.

3.3.2 Event Refinement

As mentioned above, a trial origin is passed to the Association Loop where it is developed and refined in an iterative sequence of adding new associations, relocating, and removing inconsistent associations. This is where **ESAL** can add late-arriving secondaries and any primaries that may be missing. Travel times, azimuths, and slownesses are predicted for arrivals within a window relative to the origin location and those with sufficiently small residuals are associated.

If this process were limited to non-defining secondary phases, we would expect it to be relatively fast since it would not require relocating the origin. However, it is possible that **GAassoc** could occasionally miss consistent primaries, so **ESAL** must be able add them and relocate the origin.

The Association Loop is organized in a parameterized series of discrete stages, or goals, which allow control over issues like which stations and which phaseids are considered first. Since the hypotheses provided by **GAassoc** are pretty well formed, we provide the ability to skip some of the earlier goals[3].

1.[*primary-time-weight, secondary-time-weight, array-azimuth-weight, array-slowness-weight, single-station-azimuth-weight, single-station-azimuth-weight, weighted-count-confirmation-threshold*]
2.[*ga-ordering-parameter*]
3.[*initial-ga-goal*]

When the location has stabilized (all consistent defining phases have been associated), a number of different event confirmation tests can be applied to the working origin. The working origin is abandoned if it fails any of the tests. The tests we apply are: 1) an upper limit on the size of the error ellipse[1], 2) the same weighted-count test that was applied at the end of the trial origin formation, and 3) a network probability of detection test[2]. The network probability of detection test is a new test in **ESAL**, and it is the same test that is applied during **GAassoc**. It is intended to eliminate origins with a small number of associations that were not detected at a significant number of probable stations.

1.[*semi-major-axis-threshold*]
2.[*probdet-ce-threshold, max-stations-for-probdet-ce-test, probdet-reliability-factor*]

References

Anderson, J., W. Farrell, K. Garcia, J. Given and H. Swanger, CSS Version 3 Database: Schema Reference Manual, *Tech. Rep. SAIC-90/1235*, Science Applications International Corporation, 59 pp., 1990.

Bache, T., S. Bratt, J. Given, T. Schroeder, H. Swanger and J. Wang, The Intelligent Monitoring System Version 2, *Tech. Rep. SAIC-91/1137*, Science Applications International Corporation, 93 pp., 1991.

Bache, T., S. Bratt, H. Swanger, G. Beall and F. Dashiell, Knowledge-based interpretation of seismic data in the Intelligent Monitoring System, *Bull. Seismol. Soc. Am.*, *83*, 1507-1526, 1993.

Bratt, S., G. Beall, H. Swanger, F. Dashiell and T. Bache, A knowledge-based system for automatic interpretation of seismic data to associate signals and locate events, *Tech. Rep. SAIC-91/1281*, Science Applications International Corporation, 73 pp., 1991.

Bratt, S., G. Beall, H. Swanger, F. Dashiell and T. Bache, A knowledge-based system for automatic interpretation of seismic data to associate signals and locate events, *Tech. Rep.* [in progress, revised edition of *SAIC-91/1281*], Science Applications International Corporation, 1994.

Given, J., W. Fox, J. Wang and T. Bache, The Intelligent Monitoring System: Software Integration Platform, *Tech. Rep. SAIC-93/1069*, Science Applications International Corporation, 32 pp., 1993.

Kerr, A. (ed.), Overview GSETT-3, Report prepared by the *GSE Working Group on Planning*, 9 pp., October, 1993.

Leonard, S., Automatic global event association and location estimation using a knowledge based approach to generalized beamforming, Proceedings of the 15th Annual PL/ARPA Seismic Research Symposium, *PL-TR-93-2160*, 248-255, 1993.

Ringdal, F. and T. Kværna, A multi-channel processing approach to real time network detection, phase association, and threshold monitoring, *Bull. Seismol. Soc. Am.*, *79*, 780-798, 1989.

Sereno, T. and G. Patnaik, Initial wave-type identification with neural networks and its contribution to automated processing in IMS Version 3.0, *Tech. Rep. SAIC-93/1219*, Science Applications International Corporation, 37 pp., 1993.

Swanger, H., J. Anderson, T. Sereno, J. Given and D. Williams, Extensions to the Center Version 3 Database (Rev. 1), Tech. Rep. SAIC-93/1123, Science Applications International Corporation, 106 pp., 1993.

Taylor, D. and S. Leonard, Generalized beamforming for automatic association, Proceedings of the 14th Annual PL/ARPA Seismic Research Symposium, *PL-TR-92-2210*, 422-428, 1992.

Appendix A: StaPro Parameter Descriptions

This appendix contains the **StaPro** manual page which includes a detailed description of all user-parameters.

NAME

StaPro - Station Processing

SYNOPSIS

StaPro [getpar(3) argument list]

DESCRIPTION

StaPro determines single station grouping and phase identification of seismic detections recorded at three-component and array type stations.

StaPro is written in C integrated with CLIPS and is very quick. It is run on one station at a time. StaPro can be run in a distributive environment allowing multiple stations to be processed simultaneously.

Station dependent parameters can be tuned for each station as time permits. These parameters are stored in individual parameter files allowing maximum control over every station.

The program consists of three main parts: initial wave-type, grouping and regional phase identification. The default method of determining initial wave-type is by either the neural network or a set of default rules. The neural network is used when station weights are available (see nnetweights-file) otherwise, the default rules are used. Station specific rules (SSR), written in CLIPS by the user, may be applied to supplement or override the default method (see section on Writing Station Specific Rules below). Grouping is performed for teleseismic, local and regional phases with noise detections being ignored. Phase identification for regional S phases is determined using Bayesian analysis. Phase prediction determines the remaining phases.

The processing interval and station name must be given on the command line when StaPro is run from the pipeline.

StaPro is designed to properly handle edge-effects when no adjacent data is available.

ARGUMENTS

CONTROL ARGUMENTS

sta is the station name. Required.

start-time is the processing start time. Required.

end-time is the processing end time. Default is start-time + duration.

duration is the duration of processing being requested.Default is 900 seconds (i.e., 15 minutes).

DATABASE ARGUMENTS

database is the database name to be used. The arrival, apma, amp3c, arrival-aux, sbsnr, and mag_coefs tables are read. The arrival, stassoc, assoc, origin, and origerr tables are written. A database name is required.

arrival-table specifies a non-standard arrival table name. The station, time, arid, azimuth, delta azimuth, delta time, phase, slowness, delta slowness, rectilinearity, period, amplitude and stassid are read from this table. The iphase, stassid, auth, and lddate (UTC) fields are updated when processing is completed. Also, the delaz, azimuth, delslo, and slowness is updated for three-component stations. Default is "arrival".

amp3c-table specifies a non-standard amp3c table name. The horizontal to vertical power ratios (htov) are read from this table based on center frequencies (cfreq#) specified either from the neural network weights file or by user parameters (if given). Otherwise, this table is not read. Default is "amp3c".

apma-table specifies a non-standard apma table name. The planarity (plans), short-axis incident angle (inang1), long-axis incident angle (inang3), maximum to minimum horizontal amplitude ratio (hmxmn), horizontal to vertical power ratio measured at maximum rectilinearity (hvratp#) and horizontal to vertical power ratio measured at maximum amplitude (hvrat) are read from this table. Default is "apma".

arrival-aux-table specifies a non-standard arrival-aux table name. The fkqual and fstat are read from this table. Default is "detection".

sbsnr-table specifies a non-standard sbsnr table name. The short term average amplitude (stav) and long term average amplitude (ltav) are read from sbsnr for the specified channel (see sbsnr-chan). This is the preferred amplitude. The value for ltav represents noise in the magnitude calculation. Default is "sbsnr".

sbsnr-chan specifies the channel to be used when querying the sbsnr table to obtain a short term average amplitude. If there is no preferred amplitude (stav) available then the arrival amplitude will be used. Required.

stassoc-table specifies a non-standard stassoc table name. New tuples are inserted into this table once processing is complete. Old stassids which were dissolved are deleted from this table. Default is "stassoc".

assoc-table specifies a non-standard assoc table name. Associated detections of confirmed events are written here. Old orids are deleted from this table. Default is "assoc".

origin-table specifies a non-standard origin table name. Confirmed event information is written here. Old orids are deleted from this table. Default is "origin".

origerr-table specifies a non-standard origerr table name. Location residuals for confirmed events are written here. Old orids are deleted from this table. Default is "origerr".

mag-coefs-table specifies a non-standard mag-coefs table name. Contains station dependent magnitude correction coefficients by phase. Default is "mag-coefs".

kaudit-table specifies a non-standard kaudit table name. Default is "kaudit". This table is not used at this time.

maxrec specifies the maximum number of records to be returned from the database. A warning message will be printed if more records exist in the database. This value may need to be larger when processing a long time window. Default is 500.

vendor specifies the database vendor name. Default is oracle.

CLIPS RULE FILE ARGUMENTS

default-rule-files contain the default method of determining initial wave-type (see DESCRIPTION above and WRITING STATION SPECIFIC RULES below). This is a CLIPS rules file. Default is StaPro-IWT.clp.

sta-rule-file is a CLIPS file written by the user. It is used to either supplement or override the default method of determining initial wave-type (see DESCRIPTION above). Optional.

INITIAL WAVE-TYPE ARGUMENTS

max-delslo is the maximum delta slowness. Default is 0.5 seconds/degree.

noise-fkqual-fstat determines phase to be noise if fkqual is greater than or equal to x and fstat is less than y. Default is "5.0,3.0"

min-fkqual determines phase to be noise if fkqual is greater than or equal to x. Default is 5.

max-noise-velocity determines phase to be noise if velocity is less than or equal to x. Default is 2.9 am/second.

p-s-velocity-threshold determines phase to be S if velocity is less than or equal to x. Otherwise, phase is assigned P. Default is 5.7 km/second.

min-teleseism-velocity determines phase to be T if velocity is greater than or equal to x. Default is 11.0 km/second.

min-p-rect determines phase to be P if rect is greater than or equal to x. Default is 0.7.

max-p-hvrat determines phase to be P if hvrat is less than or equal to x. Default is 1.0.

min-p-freq determines phase to be P if freq is greater than or equal to x. Otherwise, phase is assigned T. Default is 3.0 Hertz.

nnet-weights-file holds the name of the neural network weights file which has historically been called ipnnwts.tbl. This file contains weights for many stations. However, if no weights are found for the station being processed then the default rules will be implemented to determine initial wave-type. No default weights file specified. Optional.

nnet-log-file is written/overwritten each StaPro run if a filename is specified. It will contain a one line report regarding all detections processed by the neural network. No default log file specified. Optional.

site-file is filename and location of site file to be used. StaPro needs the station location and station type from this file. It should be noted that the station type in this file may be different than that found in the site database table. For example, the array ARA0 should have 'hfa' type in the site file whereas 'ss' will be found in the database table for the (3-component) station ARA0. Required.

def-3c-delaz is a default delaz to be used when the arrival.delaz is not valid. This is only for three-component stations. Default is 20.0.

def-3c-delslo is a default delslo to be used when the arrival.delslo is not valid. This is only for three-component stations. Default is 5.0.

ab-constant is a free-surface constant used in converting velocity to slowness. It's value is for a poisson solid (e.g., air over rock) and is approximated by a/(2*b*b) where "a" is compressional velocity and "b" is the S-wave velocity. Default is 0.3.

cfreq1 is the center frequency value used when measuring amplitude and is stored in the amp3c table. The value of this first cfreq has historically been 0.25. Default is -999.0.

cfreq2 is the center frequency value used when measuring amplitude and is stored in the amp3c table. The value of the second cfreq has historically been 0.50. Default is -999.0.

cfreq3 is the center frequency value used when measuring amplitude and is stored in the amp3c table. The value of the third cfreq has historically been 1.00. Default is -999.0.

cfreq4 is the center frequency value used when measuring amplitude and is stored in the amp3c table. The value of the fourth cfreq has historically been 2.00. Default is -999.0.

cfreq5 is the center frequency value used when measuring amplitude stored in the amp3c table. The value of the fifth cfreq has historically been 4.00. Default is -999.0.

PHASE GROUPING ARGUMENTS

group-first-s-p-max-time is the maximum time between first P and first S in grouping. Default is 360.0 seconds.

group-delaz-factor is used in compatibility tests in grouping. Default is 3.0.

max-grouping-time-no-azimuth is the maximum time after P that a compatible arrival allowed to exist. Used when no azimuth data available. Default is 120.0 seconds.

min-grouping-amplitude-factor is the minimum ratio of S to P amplitudes for compatibility test. Used when no azimuth data available. Default is 0.4.

max-grouping-amplitude-factor is the maximum ratio of S to P amplitudes for compatibility test. Used when no azimuth data available. Default is 25.0.

group-amplitude-tolerance is a threshold ratio for compatibility test. Used when azimuth data is available. Default is 40.0.

local-max-s-p-time determines an event to be local if the S-P time is less than x. Default is 25.0 seconds.

local-delaz-factor is used in compatibility test for local events. Default is 2.5.

local-min-p-p-separation is a time separation argument where the first P of a local event must be this many seconds from any previous, compatible P. Default is 25.0 seconds.

max-p-coda-arrival-time is the maximum time a P coda phase can exist with respect to the first P. Default is 20.0 seconds.

teleseism-group-max-width is the maximum time between first and last teleseismic phases of same group. Default is 30.0 seconds.

teleseism-azimuth-tolerance is the maximum azimuth difference between arrival and first T. Used during teleseismic compatibility test. Default is 25.0 degrees.

max-rg-time is the maximum time after P for a phase to be called Rg. Used in largest S analysis. Default is 65.0 seconds.

min-lg-frequency determines phase to be Lg if frequency is greater than or equal to x. Otherwise, phase is assigned Rg. This argument is also used in Regional Phase Identification. Default is 1.54 Hertz.

group-s-p-time-factor times S-P defines the window where additional S arrivals are considered to be compatible with the group. Default is 2.1.

REGIONAL PHASE ID ARGUMENTS

bayes-file contains Bayesian probabilities for regional S wave-types for several stations. Default is "bayes.tbl".

phase-distance-file contains limits on distance and depth for specified phases. Default is "phase-distance-ranges.txt".

tt-tables-path is the path and prefix for the travel time tables. A typical path is "/prj/shared/ops/data/tab/tab".

Required

tt-phases is a list of travel time tables (by suffix) to be loaded. Default is "'P','Pn','Pg','S','Sn','Lg','Rg'".

s-split-parm is used to define first S of two S-context for Bayesian analysis. Default is 0.4.

min-sn-p-time is used in Bayesian analysis to see if S is too close to be called Sn. Default is 60.0 seconds.

sn-confidence-threshold is the minimum Bayesian probability for the Sn phase. Default is 0.5.

regional-min-s-p-time is the S-P threshold used in Bayesian analysis to categorize context of P phase as either regional P or intermediate P. Default is 30.0 seconds.

pg-tol is the largest residual allowed between predicted and observed arrival times to be considered Pg. Default is 3.0 seconds.

lg-tol is the largest residual allowed between predicted and observed arrival times to be considered Lg. Default is 5.0 seconds.

rg-tol is the largest residual allowed between predicted and observed arrival times to be considered Rg. Default is 5.0 seconds.

lg-min-velocity is the minimum velocity of Lg. Default is 3.1 km/second.

lg-max-velocity is the maximum velocity of Lg. Default is 5.0 km/second.

rg-min-velocity is the minimum velocity of Rg. Default is 2.8 km/second.

rg-max-velocity is the maximum velocity of Rg. Default is 3.7 km/second.

EVENT CONFIRMATION ARGUMENTS

semi-major-axis-threshold is the maximum allowable value for the location error ellipse. Default is 400000.0 kilometers.

event-confirmation-threshold is the minimum weighted count for a confirmed event. Default is 2.6.

primary-time-weight is a weight applied to the number of primary time attributes during the weighted count equation. Default is 1.0.

secondary-time-weight is a weight applied to the number of secondary time attributes during the weighted count equation. Default is 0.7.

array-azimuth-weight is a weight applied to the number of azimuth attributes during the weighted count equation. This weight is for an array station. Default is 0.5.

array-slowness-weight is a weight applied to the number of slowness attributes during the weighted count equation. This weight is for an array station. Default is 0.5.

3c-azimuth-weight is a weight applied to the number of azimuth attributes during the weighted count

equation. This weight is for a three-component station. Default is 0.25.

3c-slowness-weight is a weight applied to the number of slowness attributes during the weighted count equation. This weight is for a three-component station. Default is 0.25.

DEBUG ARGUMENTS

verbose controls the level of detail of messages printed to screen regarding station processing. Values range from 0 to 5. Zero meaning less detail and three meaning more detail. Default is 1.

locator-verbose controls the level of detail of messages printed to locator-outfile-name regarding the locator (used during phase prediction). Values range from 0 to 4. Zero meaning less detail and four meaning more detail. Default is 0.

locator-outfile-name will contain locator messages constrained by locator-verbose flag. Default is "StaPro_loc.err".

auditing-level is not currently used.

log-file is not currently used.

INPUTS

DATABASE

The following tables are read: arrival, amp3c, apma, arrival-aux, sbsnr and mag-coefs.

FILES

The following files are read: bayes.tbl, phase-distance-ranges.txt, travel time tables, site file, ipnnwts.tbl, StaPro-IWT.clp, and station specific rules (if provided).

OUTPUTS

DATABASE

The following tables are written: arrival, stassoc, assoc, origin, and origerr.

FILES

The following files are written (if requested): neural network log file and locator log file.

WRITING STATION SPECIFIC RULES

SSRs are written in CLIPS. Each rule has two parts consisting of a left-hand side (LHS) where conditional elements are listed, and a right-hand side (RHS) where actions are listed. An arrow ("=>") separates the LHS from the RHS. The analogy of the common If-Then statement applies to CLIPS rules. All conditions listed on the LHS must be met before actions on the RHS can begin. All SSRs for StaPro must contain (program-state ssr-initial-wave-type) as a conditional element. This will ensure proper processing flow of the rules. In addition, at least one of the rules must contain the action which summarizes the overall completion status for the SSRs. This is done by placing (status 0|1) on the RHS. A status of 1 represents satisfactory completion whereas 0 is unsatisfactory. SSRs which determine a valid initial wave-type should assert this information on the fact list as (phase P|T|S|N). The SSRs which can not determine a particular phase can pass-on information to the default

rules to exclude a particular phase. The excluded phase should be asserted on the fact list as (not-t), (not-p), (not-s), or (not-n). See the default CLIPS rules file for examples of CLIPS rules. Also, see the CLIPS Reference Manual for details regarding rule creation.

The following are three categories of facts which are asserted on the CLIPS fact list for each detection. These facts and the functions listed below are available for use during SSRs.

CONTROL FACTS

```
iwt-not (not-p 0) (not-t 0) (not-s 0) (not-n 0)        ;; set to FALSE
program-state ssr-initial-wave-type                    ;; when SSRs given
program-state initial-wave-type                        ;; when no SSRs given
use-default-rules                                      ;; when no SSRs given
yes-wts                                  ;; when neural network weights available
no-wts                                ;; when no neural network weights available
```

RDBMS FACTS

(by table)

ARRIVAL:	station, time, arid, stassid, azimuth, delaz, slow, delslo, rect, per
APMA:	plans, inang1, inang3, hmxmn, hvratp, hvrat
AMP3C:	htov1, htov2, htov3, htov4, htov5
DETECTION:	fstat, fkqual
SITE:	statype

PARAMETER FACTS

min-p-rect

max-p-hvrat

min-p-freq

max-delslo

noise-fkqual-fstat

min-fkqual

min-teleseism-velocity

p-s-velocity-threshold

max-noise-velocity

FUNCTIONS AVAILABLE

Value Range Checkers:

The following range check functions return TRUE if ?value is in range. Otherwise, FALSE is returned.

check-rect-range (?value)

check-hvrat-range (?value)

check-per-range (?value)

check-slow-range (?value)

check-delslo-range (?value)

check-fstat-range (?value)

check-fkqual-range (?value)

Noise Screening:

Determines by fkqual and fstat if the arrival should be classified as noise. Returns TRUE or FALSE.

noise-screener (?fk ?minfk ?maxfk ?fs ?minfs)

Verbose Checker:

Checks level of verbose flag set by user parameter.

vchk4 ()

EXAMPLE:

The following is a typical par file which processes station "GAR". It accesses the nnet9 database. The start-time is specified as 679000000. The sbsnr channel is "sz0204". The directory /prj/shared/ops/data/iasp91/iasp91 holds the travel time tables. A beta site file is used for this station. StaPro will try to use the neural network provided weights exist for this station in the file ipnnwts.tbl. The Par_file follows:

```
# -----------------------
# StaPro Example Par File: GAR.par
# Execution: StaPro par=GAR.par
# -----------------------
clip_path=/home/StaPro/data
tbl_path=/home/StaPro/data/tables
tt_path=/data/tab
# ----------------------------
sta=GAR
start-time=679000000
end-time=679500000
verbose=3
max-delslo=999.9            # def=0.5
# ----------------------------
database=acount_name/password@t:machine:two_task
sbsnr-chan=sz0204
amp3c-table=amp3c
apma-table=apma
arrival-table=arrival_stapro
arrival-aux-table=detection
sbsnr-table=sbsnr
stassoc-table=stassoc_stapro
# ----------------------------
tt-table-path=$(tt_path)
```

```
bayes-file=$(tbl_path)/bayes.tbl
site-file=$(tbl_path)/beta.site
default-rule-files=$(clip_path)/StaPro-IWT.clp
phase-distance-file=$(tbl_path)/phase-distance-ranges.txt
nnet-weights-file=$(tbl_path)/ipnnwts.tbl
nnet-log-file=$(sta)_nnet.log
# ----------------------------
```

AUTHOR

Rick Jenkins
Geophysical Systems Operation
Science Applications International Corporation
San Diego, California

Appendix B: GA Subsystem Parameter Descriptions

This appendix lists descriptions of the user-parameters for **GAcons** and **GAassoc**. The user-input to both programs is through command-line arguments. These arguments can be stored in a "parameter file," and the name of the file is specified on the program command-line (e.g., *GAcons par=GAcons.par*).

GAcons User-Parameters

phases:
List of phases for which to compute grid cell - station information.

net:
Network name.

output_path:
Directory path name where output file(s) will reside.

output_prefix:
Prefix name for output file(s) containing the information for each sector. This overrides the default prefix.

nevents:
Number of events with magnitude between *min_mag* and *max_mag* to use in simulations to determine first-arrival stations. These events will have to be detected at least at *nstat_det* stations to be considered valid events for this purpose. As many events as necessary are simulated until *nevents* are detected at *nstat_det* stations.

percent:
Percentage of the total number of events simulated within a grid cell to have first arrival at station for that station to be considered a first arrival station.

nstat_det:
Number of stations required for an event to be considered valid for the simulation.

grid_spacing:
Grid spacing in degrees for the approximately uniform grid on the sphere.

min_mag:
Minimum magnitude for event simulations.

max_mag:
Maximum magnitude for event simulation.

atten_file:
Name of attenuation file, including path.

table_path:
Directory path name to travel time tables.

num_sects:
Number of sectors to divide the sphere into (for parallelization purposes). This also determines the number of files generated by **GAcons**.

vendor:
Name of the database vendor (e.g., "oracle").

database:
Name of the database where to find network information.

account:
Database account name.

maxrecs:
Maximum number of records to read from the database.

event_file:
Name of the file containing the seismicity data used in establishing the depth cells.

min_num_events_per_10sq_deg:
Minimum number of events in a 10 degree square and the depth interval corresponding to the depth cell for that cell to be taken into consideration.

depth_points:
Depth in kilometers of center of depth cells.

depth_widths:
Width in kilometers of depth cells.

dist_depth_range_file:
Name of the file containing information about the range of definition in distance and depth of seismic phases.

Sample GAcons parameter file:

```
vendor="oracle"
database="t:machine:two_task"
account="account_name/password"
output_path="/home/ga/SDG/"
maxrecs=10000
```

```
grid_spacing=3.
num_sects=1
net=GSETT3
percent=1
nevents=200
nstat_det=2
min_mag=3.
max_mag=5.
atten_file="slowamp.P"
table_path="/data/tab"
phases="Pn,PKPdf,P,Lg,S,PcP"
event_file="/home/ga/data/1980-1993.pde_depths"
min_num_events_per_10sq_deg=1
depth_points="65.0,130.0,240.0,400.0,650.0"
depth_widths="32.0, 40.0, 80.0,100.0,150.0"
dist_depth_range_file="/home/ga/data/GA_dist_depth_ranges"
```

GAassoc User-Parameters

Database interface parameters:

vendor:
Name of the database vendor (e.g., "oracle").

database:
Name of the database for input **arrival** table and output **assoc**, **origin**, **origerr** tables.

account:
Database account name.

maxrecs:
Maximum number of records to read from the database.

in-arrival-table:
Name of the input **arrival** table. The detections to be associated are read from this table. Station processing should be run on the detections before using **GAassoc**.

in-assoc-table:
Name of the Input **assoc** table. The belief field from this table is read for each arrival. It is also used to screen arrivals associated with local events with *ML* magnitude less than a user-specified value.

in-origin-table:
Name of the input **origin** table. This is used to screen local events with *ML* magnitude less than a use-specified value.

in-origerr-table:
Name of the Input **origerr** table associated with the input **origin** table.

out-origin-table:
Name of the output **origin** table. This will contain the **origin** records for the preliminary events formed by **GAassoc**.

out-assoc-table:
Name of the output **assoc** table. This table contains the **assoc** records associated with the preliminary events formed by **GAassoc**.

out-origerr-table:
Name of the output **origerr** table. This table contains the **origerr** records associated with the preliminary events formed by **GAassoc**.

net:
Name of seismic network (e.g. GSETT3).

minimum_ml_previously_determined:
Minimum local magnitude of events whose arrivals will be tentatively associated by **GAassoc**. Arrivals that have been associated by station processing with local events of magnitude less than this value are not considered for association.

input_path:
Path name to the directory where the input grid file produced by **GAcons** is located.

input_file:
Name of input grid file produced by **GAcons**. This file contains the grid information to be used by **GAassoc** in the forming of preliminary events. This file name must be specified.

table_path:
Path name to travel time and magnitude tables directory.

atten_file:
Path name and file name for the attenuation tables used for probability of detection calculations.

mb_dist_depth_suffix:
Suffix for *mb* tables.

Association loop parameters:

start_time:
Epoch time of the starting time for analysis.

end_time:
Epoch time of the end time for analysis.

lookback:
Time in seconds before the start time to include in the analysis. Events with origin time between *start_time-lookback* and *end_time-lookback* are considered in the analysis. Arrivals between *start_time-lookback* and *end_time* are read and considered in the analysis.

belief_threshold:
Threshold value for the belief field in the **arrival** table. If the belief (assigned by **StaPro**) is above this threshold then the phase identification cannot be changed by **GAassoc**.

phases:
Names of phases to be used in the association. This must be a subset of the phases used in **GAcons**.

primary_phases:
Names of phases to be considered "primary" phases.

num_first_sta:
Maximum number of "first-arrival stations" to use from the grid file produced by **GAcons**. The stations are ordered with the highest probability station first.

count_limit:
The preliminary events are examined when *count_limit* of them have been formed and redundancy analysis, event splitting and event confirmation are performed.

freeze_arrivals_at_beam_points:
If this string is present in the parameter file, arrivals will be frozen at each beam point once associated with a preliminary event. If this string is not present in the parameter file no freezing is performed.

primary_required_for_secondary:
If this string is present in the parameter file, an arrival can be associated to a preliminary event only when there is a corresponding primary phase from the same station already associated to that event.

regional_S_phases:
List of regional *S* phases. A regional *P* phase cannot be associated with a grid cell at teleseismic distance if station processing grouped it with a compatible *S* phase in the list of *regional_S_phases*.

forward_transformation_list:
For each phase identified by station processing, this list restricts the phase type that it can be transformed into by **GAassoc**.

sigma_time:
Sigma factor for time measurement uncertainty. This factor is multiplied by the deltim standard deviation to determine the interval for the preliminary screening done during the search for corroborating phases.

sigma_slowness:
Sigma factor for slowness measurement uncertainty (see the *sigma_time* description).

chi_limit:
Threshold value for the chi-square test. This is used within the association loop to determine if a corroborating arrival belongs to a preliminary event formed by a *DRIVER* arrival. A typical value is 0.99.

probdet_before_location:
If this string is present in the parameter file, a network probability test is performed prior to location.

redundancy_required:
If this string is present in the parameter file, a complete redundancy test is performed after the association loop. This is done in the normal operating mode.

Optional processes parameters:

location_required:
If this string is present, the locator module is invoked.

probdet_after_location:
If this string is present in the parameter file, a network probability test is performed after location.

residual_over_sigma_max:
Maximum value of the ratio of the residual of the network probability value to the estimated standard deviation. This parameter is used in both probability of detection tests (before and after location). A typical value is 3.

max_obs_net_prob:
Maximum number of observations above which no probability of detection test is applied. This value and the value of the *residual_over_sigma_max* parameter are used by both the pre-location and post-location probability of detection tests.

Location and confirmation module parameters:

loc_conf_level:
Locator confidence level. This is the confidence level at which the location error ellipse is computed.

loc_verbose:
Verbose value for the locator. Refer to the locator documentation for options.

loc_fix_depth:
If this string is specified, the locator keeps the depth fixed.

chi_outlier:
Chi-square threshold value used in the post-location outlier analysis within the locator module. This value is used to determine whether an arrival is an outlier for a particular event and to discard it from the preliminary event it if it is the worst outlier.

max_smajax:
Maximum permissible semi-major axis of the location error ellipse. This is one of the confirmation criteria for a preliminary event.

req_num_of_defining_detections:
Minimum number of detections for an event to be confirmed.

weight_threshold:
Minimum "weight" of an event for it to be confirmed. This is compared to the sum of all weights for the defining observations forming the event (see *primary_time_weight*, *secondary_time_weight*, *array_azimuth_weight*, *array_slow_weight*, *3comp_slow_weight*, *3comp_azimuth_weight*). This weighted-count confirmation test is described by *Bratt et al.* [1991, 1994].

primary_time_weight:
Weight assigned to arrival times for primary phases for the weighted-count event confirmation test [*Bratt et al.*, 1991, 1994].

secondary_time_weight:
Weight assigned to arrival times for secondary phases for the weighted-count event confirmation test [*Bratt et al.*, 1991, 1994].

array_azimuth_weight:
Weight assigned to array azimuths for the weighted-count event confirmation test [*Bratt et al.*, 1991, 1994].

array_slow_weight:
Weight assigned to array slowness for the weighted-count event confirmation test [*Bratt et al.*, 1991, 1994].

3comp_slow_weight:
Weight assigned to slowness from 3-component data for the weighted-count event confirmation test [*Bratt et al.*, 1991, 1994].

3comp_azimuth_weight:
Weight assigned to azimuth from 3-component data for the weighted-count event confirmation test [*Bratt et al.*, 1991, 1994].

Sample GAassoc parameter file:

```
vendor="oracle"
database="t:machine:two_task"
account="account_name/password"
maxrecs=200000
in-arrival-table="arrival"
in-assoc-table="assoc"
in-origin-table="origin"
in-origerr-table="origerr"
out-origin-table="origin_ga"
out-assoc-table="assoc_ga"
out-origerr-table="orrigerr_ga"
minimum_ml_previously_determined=2.5
atten_file="slowamp.P"
table_path="/data/tab"
mb_dist_depth_suffix="pfact"
input_path="/home/ga/SDG"
input_file="GSETT3.spacing3.sector.-180deg.to.180deg"
num_first_sta=5
count_limit=10000
forward_transformation_list="(P PKPdf Pdiff Pn S ScP PKPab PKPbc
     PP ScS),(S Rg Sn Lg),(Pn P Pg Pdiff S ScS),(Lg Sn Rg),(Sx Sn
     Lg Rg S ScS),(Tx PcP PKPbc PKPab),(Rg Lg),(Sn Lg S)"
phases="Pg,Pn,P,Pdiff,PKPdf"
primary_phases="P,PKPdf,Pn,Pg,Pdiff"
freeze_arrivals_at_beam_points
primary_required_for_secondary
regional_S_phases="Sn,Lg,Rg,Sx"
sigma_time=3.
sigma_slowness=3.
start_time=789004800.
end_time=789033600.
chi_limit=.99
#
#optional processes parameters
#
#probdet_before_location
location_required
redundancy_required
probdet_after_location
#
```

```
#location parameters
#
loc_conf_level=0.90
loc_verbose=0
loc_fix_depth
chi_outlier=.99
#
# Event confirmation criteria
#
max_smajax=500.0
residual_over_sigma_max=3.
max_obs_net_prob=10
req_num_of_defining_detections=3
weight_threshold=3.9
primary_time_weight=1.0
secondary_time_weight=1.0
array_azimuth_weight=0.25
array_slow_weight=0.25
3comp_slow_weight=0.0
3comp_azimuth_weight=0.0
```

Appendix C: EServer/ESAL Parameter Descriptions

Most of the details of **ESAL** processing are controlled by configuration parameters set at run-time. In this section we list the parameters that have been added to allow processing of **GAassoc** hypotheses with a brief definition. The convention is:

PARAMETER-NAME:
Definition.

GA-ORIGIN-FILENAME:
This specifies the file name for origin data (i.e., origin relation) for GA preliminary events. It will be sought in the directory formed by the concatenation of *DATA-PATHNAME* and *DETECTION-PATHNAME*. The value must be a string.

GA-ORIGERR-FILENAME:
This specifies the file name for origin error data (i.e., origerr relation) for GA preliminary events. It will be looked for in the directory formed by the concatenation of *DATA-PATHNAME* and *DETECTION-PATHNAME*. The value must be a string.

GA-ASSOC-FILENAME:
This specifies the file name for association data (i.e., assoc relation) for GA preliminary events. It will be sought in the directory formed by the concatenation of *DATA-PATHNAME* and *DETECTION-PATHNAME*. The value must be a string.

GA-CONFLICT-RESOLUTION-HEURISTIC:
This specifies which heuristic to use for resolving multiple-association conflicts during GA trial origin formation. If the heuristic is *ERROR-ELLIPSE-AREA*, a phase will be associated with the preliminary event with the smallest error-ellipse area. If the heuristic is *NUMBER-OF-DEFINING-PHASES*, a phase will be associated with the preliminary event with the largest number of defining phases; ties are broken by the smaller error-ellipse area. If the heuristic is *REVISED*, conflicts are resolved based on an ordered list of criteria including: defining vs. non-defining, firm-association vs. weak-association, number of defining-components, and error-ellipse area; this is essentially the same heuristic used during regular conflict resolution if the selected heuristic is *REVISED* (cf. *CONFLICT-RESOLUTION-HEURISTIC*). If the heuristic is *REVISED-NDEF*, a phase will associated with the preliminary event with the largest number of defining phases _but_ the conflict resolution will be ordered starting with the best preliminary event and each loser of a conflict resolution will have its number of defining phases revised before the next conflict is resolved. The value must be one of: *ERROR-ELLIPSE-AREA, NUMBER-OF-DEFINING-PHASES, REVISED, REVISED-NDEF.*

GA-ORDERING-PARAMETER:
This specifies the parameter that will be used to order the selection of GA preliminary events as trial origins. The value must be one of: *ORIGIN-TIME, NUMBER-OF-DEFINING-DETECTIONS, BODY-MAGNITUDE.*

INITIAL-GA-GOAL:
This index specifies the first goal out of *GOAL-ORDER* to be used in the construction of GA working origins. The value should be a number between 1 and the length of *GOAL-ORDER*. This allows GA preliminary events to be treated as partially formed and not have to go through the early goals. The value must be an integer.

MAX-STATIONS-FOR-PROBDET-CE-TEST:
The probdet network-likelihood event confirmation test is only applied to events detected at fewer than this number of stations. (cf. *PROBDET-CE-THRESHOLD*). The value must be an integer.

PROBDET-CE-THRESHOLD:
An event will be confirmed only if the probdet network-likelihood is close enough to its expected value. Specifically, $(A\text{-}B)/C \leq threshold$, where B is the log of the network likelihood, A is its expected value, C is the square root of the variance of the expected value, and *threshold* is this parameter. The log of the network likelihood is: Σ(k|detecting stations) log $p(k)$ + Σ(k|non-detecting stations) log $(1\text{-}p(k))$, where $p(k)$ is the probability of detection times the station reliability times *PROBDET-RELIABILITY-FACTOR*. This test is only applied to events detected at fewer than *MAX-STATIONS-FOR-PROBDET-CE-TEST* stations. The value must be a number.

PROBDET-RELIABILITY-FACTOR:
This specifies a factor which multiplies the station reliability used in the probdet network-likelihood event confirmation test. (cf. *PROBDET-CE-THRESHOLD*). The value must be a number.

New **EServer** parameters for use with GA input:

write_ga_files:
Causes **EServer** to include GA origin, origerr, and assoc files as esal input.
Default = *nowrite_ga_files*

ga_origin:
Specifies the name of the table to use for GA origin.
Default = ga_origin.

ga_origerr:
Specifies the name of the table to use for GA origerr.
Default = ga_origerr.

ga_assoc:
Specifies the name of the table to use for GA assoc.
Default = ga_assoc.

ga_arrival:
Specifies the name of the table to use for GA arrivals.
Default = arrival.

Distribution List

AFTAC/TTR	Technical Report (2 copies)
AFTAC/TTS	Technical Report (1 copy)
CA/STINFO	Technical Report (2 copies)